

Principled design of Evolutionary Learning Systems for Large Scale Data Mining

María Auxiliadora Franco Gaviria

Thesis submitted to The University of Nottingham
for the degree of Doctor of Philosophy

July 2013

ALL MISSING PAGES ARE BLANK

IN

ORIGINAL

Dedicated to my mother, Ninfa

Abstract

Currently, the data mining and machine learning fields are facing new challenges because of the amount of information that is collected and needs processing. Many sophisticated learning approaches cannot simply cope with large and complex domains, because of the unmanageable execution times or the loss of prediction and generality capacities that occurs when the domains become more complex. Therefore, to cope with the volumes of information of the current real-world problems there is a need to push forward the boundaries of sophisticated data mining techniques.

This thesis is focused on improving the efficiency of Evolutionary Learning systems in large scale domains. Specifically the objective of this thesis is improving the efficiency of the Bioinformatic Hierarchical Evolutionary Learning (BioHEL) system, a system designed with the purpose of handling large domains. This is a classifier system that uses an Iterative Rule Learning approach to generate a set of rules one by one using consecutive Genetic Algorithms. This system have shown to be very competitive so far in large and complex domains. In particular, BioHEL has obtained very important results when solving protein structure prediction problems and has won related merits, such as being placed among the best algorithms for this purpose at the Critical Assessment of Techniques for Protein Structure Prediction (CASP) in 2008 and 2010, and winning the bronze medal at the HUMIES Awards for Human-competitive results in 2007. However, there is still a need to analyse this system in a principled way to determine how the current mechanisms work together to solve larger domains and determine the aspects of the system that can be improved towards this aim.

To fulfil the objective of this thesis, the work is divided in two parts. In the first part of the thesis exhaustive experimentation was carried out to determine ways in which the system could be improved. From this exhaustive analysis three main weaknesses are pointed out: a) the problem-dependancy of parameters in BioHEL's fitness function, which results in having a system difficult to set up and which requires an extensive preliminary experimentation to determine the adequate values for these parameters; b) the execution time of the learning process, which at the moment does not use any parallelisation techniques and depends on the size of the training sets; and c) the lack of global supervision over the generated solutions which comes from the usage of the Iterative Rule Learning paradigm and produces larger rule sets in which there is no guarantee of minimality or maximal generality.

The second part of the thesis is focused on tackling each one of the weaknesses abovementioned to have a system capable of handling larger domains. First a heuristic approach to

set parameters within BioHEL's fitness function is developed. Second a new parallel evaluation process that runs on General Purpose Graphic Processing Units was developed. Finally, post-processing operators to tackle the generality and cardinality of the generated solutions are proposed. By means of these enhancements we managed to improve the BioHEL system to reduce both the learning and the preliminary experimentation time, increase the generality of the final solutions and make the system more accessible for end-users. Moreover, as the techniques discussed in this thesis can be easily extended to other Evolutionary Learning systems we consider them important additions to the research in this field towards tackling large scale domains.

Acknowledgements

First of all, I would like to thank God for putting me in the right path always. Moreover, I would like to thank my parents, Carlos Mario and Ninfa, for their support and for giving me the opportunity to reach my goals no matter how crazy they might seem. I would also like to give thanks to the rest of the members of my family and especially to my brother Juan Carlos, for his company, support and interest in this particular research topic.

I would like to thank my supervisors Dr. Jaume Bacardit and Prof. Natalio Krasnogor, for their advice and for guiding my work during these four years. Without their guidance it would have been impossible to reach the current quality of this work. Additionally, I would like to thank my former supervisor in Venezuela, Carolina Martinez, for introducing me into this line of research and encouraging me to the PhD in this particular area in the first place.

I would also like to thank Dr. Sam Allen and Dr. Paweł Widera for their insightful suggestions about thesis writing which made this process a lot easier. In general I would like to thank my friends from ICOS and Computer Science in Nottingham, for hearing me nag about the PhD every so often and for encouraging me to see the bright side of things when things did not go as expected.

Finally, I would like to thank my friends back in Venezuela for encouraging me on taking on this adventure and being by my side even from the distance.

Gracias a todos

Contents

1	Introduction	1
1.1	Background and Motivation	1
1.2	Aims and Scope	3
1.3	Structure of the Thesis	4
1.4	Published Work	5
2	Background Work	7
2.1	Machine Learning and Data Mining	7
2.1.1	Data mining for large scale datasets	10
2.2	Evolutionary Computation and Genetic Algorithms	11
2.3	Evolutionary Learning	12
2.3.1	Components of Evolutionary Learning	12
2.3.2	Paradigms of Rule-based Evolutionary Learning	14
2.3.3	Problem domains	18
2.3.4	Knowledge representation	19
2.3.5	Efficiency enhancement methods	21
2.4	GAssist and BioHEL	25
2.4.1	GAssist	25
2.4.2	BioHEL	29
3	Methodological aspects	35
3.1	Statistical tools and performance measures	35
3.1.1	Wilcoxon signed-ranked test	36
3.1.2	Friedman test	36
3.1.3	Kruskal-Wallis test	37
3.2	Datasets	38
3.2.1	Real-world datasets	38
3.2.2	Synthetic datasets	40
3.3	Basic parameter configuration for BioHEL	41
4	Pittsburgh vs. IRL: Identifying future challenges	43
4.1	Introduction	43
4.2	Previous comparisons between GBML systems	44
4.3	Parameter sensitivity analysis vs. self-adaptation	45
4.4	Experimental design	45

4.4.1	Evaluated scenarios	47
4.4.2	Performance metrics and statistical analysis	47
4.5	Results	49
4.5.1	Finding a global standard configuration	49
4.5.2	Comparison with the best parameters per problem	52
4.5.3	GAssist vs. BioHEL	56
4.5.4	Parameter-sensitivity analysis over large scale datasets	62
4.5.5	Analysis of the ILAS windowing scheme	66
4.5.6	Comparison against other ML techniques	66
4.6	Conclusions and Further Work	70
5	Parameter impact in BioHEL's fitness function	75
5.1	Introduction	75
5.2	k-DNF boolean functions	76
5.2.1	Dimensions of difficulty	76
5.2.2	k-DNF as a Machine Learning benchmark	78
5.2.3	Previous Machine Learning studies using k-DNF functions	78
5.3	Parameter sensitivity analysis	78
5.3.1	Experimental design	80
5.3.2	GA iterations and execution time	81
5.3.3	Learning and overgeneralisation	85
5.4	Towards a rationalisation of the coverage breakpoint settings	88
5.4.1	A general coverage breakpoint setting for binary domains	92
5.4.2	Coverage breakpoint setting policies for more complex domains	92
5.5	Conclusions	93
5.6	Future research directions	94
6	Modelling the Initialisation Stage of BioHEL	95
6.1	Introduction	95
6.2	Evolutionary Learning modelling methodologies	96
6.2.1	Facetwise analysis of the initialisation stage	97
6.3	Probabilistic models for ALKR+GABIL	99
6.3.1	Schema Bound: Probability of generating a representative	99
6.3.2	Covering bound: Probability of matching an instance	103
6.3.3	Model validation	104
6.4	Towards generalised models for χ -ary attributes	106
6.4.1	Schema bound	107
6.4.2	Covering bound	109
6.4.3	Model validation	109
6.5	Ensuring a good initial population in BioHEL	111
6.6	Using the models to determine the problem structure	114
6.7	Conclusions and further work	114
6.8	Further work	115
7	Automatic theory-based adjustment of the coverage breakpoint	117
7.1	Introduction	117

7.2	Parameter setting and parameter control methodologies	118
7.3	Automatic parameter setting of the coverage breakpoint	120
7.3.1	Classification criteria	121
7.3.2	How to classify the problems?	125
7.3.3	Parameter setting procedure step-by-step	126
7.4	Experimental design and results	126
7.4.1	Analysis of the parameter setting approach over binary problems	128
7.4.2	Analysis over a real-world PSP problems	132
7.5	Conclusions and Further Work	138
8	Fast evaluation process using GPGPUs	141
8.1	Introduction	141
8.2	GPGPUs and CUDA	142
8.2.1	GPGPUs in Machine Learning	144
8.3	CUDA-based efficient fitness computation process	144
8.3.1	Coarse-grained parallel approach	146
8.3.2	Fine-grained parallel approach	149
8.3.3	Handling the ILAS windowing scheme	153
8.3.4	Integration with the learning algorithm	154
8.3.5	Handling GPGPUs with small global memory	154
8.4	Experimental design	155
8.5	Performance of the evaluation process	156
8.6	BioHEL using CUDA-based evaluation	162
8.7	CUDA execution time models	164
8.8	Conclusions	168
8.9	Further work	169
9	Post-processing the final rule sets	171
9.1	Introduction	171
9.2	Related work: Rule optimisation algorithms	172
9.3	Post-processing operators for BioHEL	174
9.3.1	Rule Pruning	174
9.3.2	Rule Cleaning	175
9.3.3	Rule Swapping	177
9.4	Experimental design	179
9.5	Results	179
9.5.1	Independent operator analysis	179
9.5.2	Interactions between CL and PR	182
9.5.3	Interactions of rule operators with SW	182
9.5.4	Statistical evaluation of the operators	182
9.6	Conclusions	184
9.7	Further work	185
10	Summary, Conclusions and Further Work	187
10.1	Summary of the thesis	187
10.2	Contributions to the area of Evolutionary Learning	190

10.3 Further work	190
10.3.1 Further research directions on BioHEL's experimentation	190
10.3.2 Further research directions on modelling the behaviour of BioHEL	191
10.3.3 Further work on setting the coverage breakpoint automatically	191
10.3.4 Further work on speeding up the evaluation process	192
10.3.5 Further work on refining the final solutions	192
10.3.6 Where to go from here?	192
A Results of the parameter sensitivity analysis of BioHEL and GAssist	195
A.1 Complete results of the parameter sensitivity analysis for BioHEL	195
A.2 Complete results of the parameter sensitivity analysis for GAssist	195
References	195

List of Figures

2.1	Example of decision tree for the contact lens prescription problem	8
2.2	Example of rule set for the contact lens prescription problem	9
2.3	Example of a linear models that separates the data in two classes and a neural network	9
2.4	Flow of Michigan XCS	16
2.5	Diagram of the fields of research presented in this thesis	19
2.6	Example of the ILAS windowing partition and window usage.	28
2.7	Representation of a classifier using ALKR in BioHEL	30
2.8	Coverage term $COV(a)$ according to rule coverage	32
3.1	Multiplexer with 8 inputs (string bits) and 3 select lines (address bits).	40
4.1	Summary of parameters analysed per system and their corresponding results sections.	46
4.2	Relative test accuracy of BioHEL according to GAssist in the small problems ordered by the number of classes	59
4.3	Relative test accuracy of BioHEL according to GAssist in the small problems ordered by the percentage of continuos attributes	60
4.4	Relative test accuracy of BioHEL according to GAssist in the small problems ordered by the number of instances	60
4.5	Relative test accuracy (considering best value observed) and number of rules obtained with different parameter configurations in large datasets	63
4.6	Relative test accuracy (considering best value observed), number of rules and execution time obtained with different window sizes in large datasets	65
4.7	Clustering of the different ML algorithms using Nemenyi critical distance	68
4.8	Execution time of the different ML algorithms over large problems	69
4.9	Algorithm selection scheme and recommended configurations for GAssist and BioHEL depending on the characteristics of the problem instances	72
5.1	Probability of having a negative example in a k-DNF function according to the number of attributes k and the number of terms r	77
5.2	Example of a randomly generated k-DNF dataset with $d = 5$, $k = 2$ and $r = 3$	79
5.3	Example of a correct and an overgeneralized rule set which solve problem in Figure 5.2	80

5.4	Number of iterations necessary to learn a good rule according to the number of attributes k and the number of terms r	82
5.5	Average execution time to learn a k-DNF function with different coverage breakpoints	84
5.6	Learning and overgeneralisation maps using various k-DNF problems and coverage breakpoints (0.1, 0.01, 0.001 and 0.0001)	86
5.7	Learning and overgeneralisation maps aligned by coverage breakpoint values using various k-DNF problems, default class 0 and coverage breakpoints (0.1, 0.01, 0.001 and 0.0001)	87
5.8	Learning and overgeneralisation map using various k-DNF problems and coverage breakpoints, where the coverage breakpoint is $1/2^t$ and the default class is 0	89
5.9	Learning and overgeneralisation map using various k-DNF problems and coverage breakpoints, where the coverage breakpoint is $1/2^t$ and the default class is the majority class.	90
6.1	Percentage of examples covered by a rule in relation with the total percentage of examples covered	102
6.2	Validation of the probability of a representative using the multiplexer problem.	104
6.3	Validation of the probability of a representative using the k-DNF problem with one rule	105
6.4	Validation of the probability of a representative for k-DNF problem with rule overlap and $p = 0.75$	106
6.5	Validation of the probability of a representative for k-DNF problem with rule overlap and $p = 0.25$	107
6.6	Validation of the probability of match using the multiplexer problems.	108
6.7	Probability of generating the schema attributes against the probability of generating a matchable rule with $e = 1$ and $d = 10$	108
6.8	Validation of the probability of a representative using the ternary multiplexer problem.	110
6.9	Validation of the probability of match using the ternary multiplexer problems	110
6.10	Population size necessary to ensure $P(rep\ exists) > 0.95$ and schema bound probabilities for the different BioHEL configurations in a binary domain $t = 2$ $e = 1$, $d = 10$ and $r = 10$	112
6.11	Population size necessary to ensure $P(cover) > 0.95$ and covering probabilities for the different BioHEL configurations in a binary domain $t = 2$ $e = 1$	113
6.12	Minimum population size to ensure a good initial population according to p with problems with $k = \{3, 5\}$, $d = \{10, 20\}$ and $r = 10$	114
6.13	Impact of t and e in the schema supply probability for $d = 10$ and $r = 10$ for BioHEL when using covering and default rule.	115
7.1	General diagram of the usage of the heuristic to determine the structure of the problem.	121
7.2	Probability of generating a representative with different values of p in a problem with $d = 10$ and $ExpAtts = 15$	123

7.3	Steps to find the adequate coverage breakpoint with and example of the final score grid.	127
7.4	Final score grids in a problem with $d = 20$, $k = 5$ and $r = 20$ for the 4 levels of noise	130
7.5	Number of additional evaluations performed to determine the coverage breakpoint value in each scenario	133
7.6	Frequency of usage of the different p values to determine the coverage breakpoint value	134
7.7	Probability of success and number of additional evaluations using different number of representatives R over k-DNF problems with $d=20$, $k=5$, $r=20$ and $minAcc=1.0$	135
7.8	Average accuracy of the best classifier during the 50 iterations of the GA using different values of p	136
8.1	Example of coalesced and non-coalesced memory access.	143
8.2	Stages of the CUDA evaluation in each iteration of the GA	146
8.3	Distribution of the global memory by the end of the first kernel function	147
8.4	CUDA parallel reduction algorithm.	148
8.5	Complete flow of data for the coarse-grained parallel approach.	151
8.6	Example of reorganisation of an instance in device memory	151
8.7	Example of reorganisation of a classifier in device memory	152
8.8	Speedup against the serial algorithm without using windowing of the different parallelisation approaches ran on different architectures	158
8.10	Largest speedup found for each one of the implementations and architectures tested depending on the number of attributes and training set size	161
8.12	Execution time model of the match kernel for different types of attributes	165
8.13	Execution time model of the instance copy operations	165
8.14	Model validation over the data obtained in the previous sections for the coarse-grained approach over the Tesla C2070	166
8.15	Model validation over the data obtained in the previous sections for the coarse-grained approach over the Tesla C1060	167
8.16	Model validation over the mixed problems for the coarse-grained approach over the Tesla C2070	168
8.17	Model validation over the mixed problems for the coarse-grained approach over the Tesla C1060	169
9.1	Example of good rules and over-specific rules for a particular problem with 3 attributes.	174
9.2	Example of rule where it is not possible to erase irrelevant attributes because of the over-generalisation of the relevant ones in a nominal problem with 3 attributes.	175
9.3	Example of different cleaning policies CL and CL2 for continuous and discrete attributes	175
9.4	Example case where rule swapping can be beneficial since it allows erasing extra rules	177
9.5	Results of the application of the different post-processing operators independently	181

9.6	Results of the application of the different combinations (in terms of order of application) of post-processing operators CL/CL2 and PR	183
9.7	Results of the application of the different combinations (in terms of order of application) of post-processing operators with the SW operator	184

List of Tables

3.1	Characteristics and classification of the datasets used in this thesis	39
3.2	Fixed parameters for the BioHEL system	41
4.1	Datasets used to compare GAssist and BioHEL and its classification	47
4.2	Parameter configuration for GAssist and BioHEL	48
4.3	Average rankings and p-values of the Friedman test to determine the best number of iterations per default class policy in GAssist	50
4.4	Average rankings and p-values of the Friedman test to determine the best default class per number of iterations in GAssist	50
4.5	Average rankings of the Friedman test over all the configurations in GAssist . . .	50
4.6	Average rankings and p-values of the Friedman test to determine the best coverage breakpoint per default class policy in BioHEL	51
4.7	Average rankings and p-values of the Friedman test to determine the best default class for each coverage breakpoint in BioHEL	51
4.8	Average rankings of the Friedman test over all the configurations in BioHEL . . .	52
4.9	Best test accuracy in GAssist using different default class policies	54
4.10	Frequency in which each one of the number of iterations in GAssist produced the best results using different default class policies	54
4.11	Best test accuracy in BioHEL using different default class policies	55
4.12	Frequency in which each one of the coverage breakpoints produced the best results in BioHEL using different default class policies	56
4.13	Comparison of the best test accuracy found per problem with the test accuracy using the default configuration for both systems	57
4.14	Comparison of the test accuracy and training accuracy of both systems	58
4.15	Comparison of the test accuracy and number of rules of both systems	59
4.16	Speedup of the BioHEL system over GAssist for the different configurations . . .	61
4.17	Test accuracy in BioHEL using different coverage breakpoints over large problems	62
4.18	Test accuracy in GAssist using different number of iterations over large problems	62
4.19	Test accuracy in BioHEL using different number of windows	64
4.20	Test accuracy in GAssist using different number of windows	64
4.21	Comparison of GAssist and BioHEL with other ML algorithms in terms of test accuracy	67
4.22	Comparison of GAssist and BioHEL with other ML algorithms in terms of execution time (s)	70

5.1	List of BioHEL's parameters different from the default configuration for the parameter sensitivity analysis	81
5.2	Model of the number of iterations necessary to learn a good rule	82
5.3	Average number of GA iterations necessary to learn a k-DNF term using different coverage breakpoints and default class 0	83
5.4	Percentage of positive examples for the last scenario solved and the first scenario not solved in BioHEL according to Figure 5.8.	89
7.1	Parameters for the heuristic used to characterise and find the coverage breakpoint for k-DNF problems.	128
7.2	Probability of success in k-DNF problems with different amounts of output noise and <i>minAcc</i> values	129
7.3	Results of the Friedman test performed to determine the best <i>minAcc</i> value depending on the noise	131
7.4	Parameters for the BioHEL system different than its fixed configuration.	135
7.5	Results over the binary <i>CN-bin</i> problem using fixed coverage breakpoints and different values of <i>p</i>	137
7.6	P-values of the Wilcoxon pairwise test to determine significant differences between the usage of different coverage breakpoints of the type 2^{-k} in the binary <i>CN-bin</i> problem	138
7.7	Performance of the heuristic over the binary <i>CN-bin</i> problem.	138
8.1	Datasets used for testing GPGPU-based fitness computation	155
8.2	Execution time in seconds of the evaluation process of the serial version and both CUDA fitness functions using 1 window	156
8.3	Speedup of the CUDA evaluation using the coarse-grained approach (on the C1060 and C2070) against the serial version	157
8.4	Speedup of the CUDA evaluation using the fine-grained approach (on the C2070) against the serial version	159
8.5	Execution time (s) of the integration of the CUDA fitness function using the coarse-grained approach in BioHEL and the serial BioHEL version	162
8.6	Speedup of the BioHEL system using the CUDA evaluation over the serial version	163
8.7	Table of fitted models for the match and the instance copy processes	164
9.1	Test accuracy, test accuracy after ensembles, number of rules and number of attributes of the rule sets before post-processing	180
9.2	Execution time of the application of each one of the different operators independently	182
9.3	Rankings of the Friedman statistical tests to compare the combinations of post-processing operators	185
A.1	Test accuracy in BioHEL using different coverage breakpoints and different default class policies	196
A.2	Train accuracy in BioHEL using different coverage breakpoints and different default class policies	197

A.3	Number of rules generated in BioHEL using different coverage breakpoints and different default class policies	198
A.4	Test accuracy in GAssist using different number of iterations and different default class policies	199
A.5	Train accuracy in GAssist using different number of iterations and different default class policies	200
A.6	Number of rules generated in GAssist using different number of iterations and different default class policies	201

Abbreviations

ADI	Adaptive Discretisation Intervals
ALKR	Attribute List Knowledge Representation
BioHEL	Bioinformatics-oriented Hierarchical Evolutionary Learning
BOA	Bayesian Optimisation Algorithm
CASP	Critical Assessment of techniques for protein Structure Prediction
CPU	Central Processing Unit
CRA	Compact Ruleset Algorithm
CUDA	Computer Unified Device Architecture
DAS	Data Analytics Supercomputer
DSS	Dynamic Subset Selection
EA	Evolutionary Algorithm
ECGA	Extended Compact Genetic Algorithm
ECL	Enterprise Control Language
EDA	Estimation of Distribution Algorithm
EL	Evolutionary Learning
EP	Evolutionary Programming
ES	Evolutionary Strategies
GA	Genetic Algorithm
GAssist	Genetic clASSIfier sySTem
GBML	Genetics-based Machine Learning
GECCO	Genetic and Evolutionary Computation Conference
GPGPU	General Purpose Graphic Processor Unit
GP	Genetic Programming
HIDER	Hierarchical Decision Rules
HPCC	High Performance Computing Cluster
IBk	k-Nearest Neighbour Algorithm

ILAS	Incremental Learning with Alternative Strata
IRL	Iterative Rule Learning
k-DNF	k-Disjunctive Normal Form
LCS	Learning Classifier System
MA	Memetic Algorithm
MDL	Minimum Description Length
ML	Machine Learning
MOEA	Multi-Objective Evolutionary Algorithm
MPI	Message Passing Interface
MPLCS	Memetic Pittsburgh Learning Classifier System
NSGA-II	Non-dominated Sorting Genetic Algorithm-II
PSP	Protein Structure Prediction
RL	Reinforcement Learning
RSS	Random Subset Selection
SIMD	Single Instruction Multiple Data
SMO	Sequential Minimal Optimisation
SVM	Support Vector Machine
UCI	University of California at Irvine
UCS	sUpervised Classifier System
XCS	eXtended Classifier System
XCSF	XCS for Function approximation
ZCS	Zeroth-level Classifier System

CHAPTER 1

Introduction

We are living in the “data deluge” era and the data processing capacity is falling far behind the data collection rates. Therefore, improving the current data mining techniques to work with larger volumes of data becomes crucial. This thesis is focused on improving Evolutionary Learning algorithms to perform data mining over large data. Particularly we focus on improving the efficiency of the BioHEL system by means of setting parameters automatically to reduce the preliminary experimentation, applying parallelisation techniques and improving the generality of the final solutions obtained.

This chapter presents in greater detail the motivations for this thesis. Afterwards, we present the aims, scope and contributions of this work. Finally, we present the roadmap for the rest of the thesis and the list of publications related to each chapter.

1.1 Background and Motivation

The data mining research field, which is focused on automatically extracting knowledge or abstractions from a set of raw data [Witten and Frank, 2005], is facing an enormous challenge: the dizzying speed at which data is generated nowadays. With the decrease of the storage costs and the developments in technology and communications in the last two decades, data is now obtained and stored from almost anywhere (internet, medical records, biotechnology, industrial records, telecommunications, military, etc.) [TheEconomist, 2010a]. For instance, a lot of our activities as a society are being recorded in different databases, from enrolling on a particular university course, to having a doctor’s appointment or simply clicking a search result over the Internet. According to [TheEconomist, 2010b] humankind created 1200 exabytes (billion gigabytes) of data in 2010. Important insights are hidden within this vast amount of data that we are not yet capable of processing.

The main problem is that the data collection rate easily exceeds the data processing rate. This is because the large scale data is often too big and complex to be mined by the available techniques, and even if sometimes, it is possible to solve these problems, the approaches used either take an undesirable amount of time or are very simple. Therefore, there is a general need to develop more powerful data mining mechanisms specially designed to work with larger data that scale well and also produce accurate and interpretable results.

But what does “large scale data” mean? There are different dimensions in which a problem can be “large and complex”. The first dimension would be the amount of registers or examples to learn from. According to [Lin, 2011] data used for machine learning at Google in 2011 contained hundreds of billions of instances. The second dimension would be the number of features analysed. For instance, some bioinformatics problems contain tens of thousands of variables to analyse [Bacardit and Llorà, 2012]. Other dimensions of difficulty would be the class imbalance, number of classes, noise, among others. For example fraud detection datasets contain a very

small fraction of positive examples, and this misrepresented class is actually the important one to classify. In this thesis we explore the two first dimensions of difficulty presented, which are the number of instances or registers available and the number of attributes or features in the problem.

Particularly this thesis is focused on the application of Evolutionary Learning (EL) to large scale data mining. EL is the application of a Genetic Algorithm (GA) [Goldberg, 1989; Holland, 1975] to extract patterns or rules from data. The GA acts upon a population of individuals (encoded solutions or parts of the solution), evolving it through time using the Darwinian selection principles. New individuals are generated through crossover and mutation and while the population grows the individuals with lower fitness are deleted to give room to better ones. When EL is used on its own to solve ML tasks (rather than in combinations with other methods) it is possible to distinguish three learning paradigms. The first one is the Michigan approach [Holland, 1975] in which the individuals of the population are independent rules which cooperate to solve the problem and are generated by a combination of reinforcement learning and genetic operators. The second one is the Pittsburgh approach [Smith, 1980, 1983] in which the individuals represent a complete solution to the problem and new solutions are generated recombining the existing ones. Finally, the last one is Iterative Rule Learning (IRL) [Venturini, 1993], which consists in learning one rule¹ at a time by means of consecutive GAs that work on the subset of instances that have not been classified by the learned rules so far.

These algorithms have proven to be good candidates to solve large scale domains, because of its intrinsic parallel nature (from the usage of a GA) and the interpretability power of the generated solutions (which are often represented by sets of rules). But still there are challenges these algorithms face when trying to handle large scale data [Bacardit and Llorà, 2012], such as the execution time or the difficulty of finding complex inner problem structures. Therefore, there are two general aspects in which the systems can be improved: efficiency and accuracy. Tangentially to these two, the interpretability of the solutions is another aspect that can be improved by trying to generate solutions that are general and easy to understand. To improve the efficiency of these algorithms different approaches have been followed so far, such as parallelisation [Cantú-Paz, 2000; Zaki and Ho, 2000], pre-processing and data selection techniques [Derrac et al., 2010; García et al., 2012], windowing techniques [Bacardit, 2004; Ishibuchi et al., 2012; Song et al., 2005], novel representations [Bacardit and Krasnogor, 2009a; Butz et al., 2008a], data intensive computing [Llorà et al., 2010], among others. Moreover, to improve the accuracy of the algorithms in more complex domains, other authors have tried to understand the theoretical limitations of the application of the GAs in the learning process [Butz, 2006; Goldberg, 2002; Orriols-Puig, 2008] to improve the algorithms in principled ways.

However, with the increase of the volume of data, there is still a gap between the state-of-the-art EL algorithms and real-world complex domains. For example, some of the largest datasets solved by EL systems have hundreds of thousands of registers [Bacardit et al., 2009b, 2012; Marín-Blázquez and Martínez Pérez, 2008; Shafi and Abbass, 2009; Song et al., 2005]. Moreover, EL systems have also been capable of solving datasets with very large numbers of features (≈ 14000 attributes) [Bacardit and Krasnogor, 2009a; Bassel et al., 2011]. However, as it was mentioned before, there exist larger real-world problems that still fall out of the domains of competence of the evolutionary computation approaches.

This thesis is focused on improving the efficiency of a particular EL system called BioHEL (Bioinformatics-oriented hierarchical evolutionary learning) system [Bacardit et al., 2009a], which was specifically designed for mining large datasets. This system is the successor of a Pittsburgh Learning Classifier System, GAssist [Bacardit, 2004], because it shares many mechanisms with the latter. The main difference between them is the change in the learning paradigm from Pittsburgh to IRL, which allows the system to tackle problems that require more complex solutions (larger rule sets), but sacrificing the global perspective of the rule set that the Pittsburgh approach provides.

¹Each individual of the population corresponds to one rule.

The BioHEL system has obtained very good results and it has been shown to be competitive against other machine learning algorithms [Bacardit et al., 2009b, 2012; Bassel et al., 2011; Glaab et al., 2012; Stout et al., 2008, 2009], but still there are some aspects in which this system could be further analysed and improved:

- An exhaustive experimentation with the BioHEL system is still lacking to determine its domains of competence, strengths and weaknesses and how the behaviours of the inherited mechanisms vary with the change of the learning paradigm.
- To apply generality pressure, BioHEL's fitness function depends on a parameter called *coverage breakpoint*, which determines how general (or extensive in terms of search space) the rules should be. This parameter has helped BioHEL to find good solutions in complex domains so far, but an extensive experimentation is still lacking to determine how this parameter should be correctly set. This parameter is usually determined through a preliminary experimentation process that may take a large amount of time.
- A formal theory is also missing to explain how the BioHEL system works. This means understanding theoretically the limitations of the system and the type of datasets over which this system is expected to work correctly. Moreover, this theory should explain how to parameterise the system to ensure learning according to the characteristics of the problem.
- Parallelisation techniques have not yet been applied to speed up the main bottleneck of the algorithm: the evaluation process.
- Since this algorithm applies IRL, the rules are generated independently from each other and there is no guarantee that the subset of rules generated is minimal or as general as possible. Mechanisms that address this issue are still needed.

1.2 Aims and Scope

The general aim of this dissertation is to improve the competence of the BioHEL system in large scale domains. The specific goals are: a) improving the efficiency of the BioHEL system in terms of execution time (including the time spent in preliminary experiments), b) establishing a methodology to set the coverage breakpoint correctly, c) develop a mechanism to set up the coverage breakpoint automatically to avoid extensive parameter tuning experimentation, d) improve the generality of the final rule sets obtained using IRL and e) present methodologies that are not only applicable to the BioHEL system but to other EL systems of similar nature.

To fulfil these goals, an exhaustive analysis is performed over the BioHEL system. First to understand further how the system works and how it should be parameterised, and then to establish how it should be improved. This analysis has two facets, an empirical and a theoretical one. The empirical facet is based on experimenting with both real-world and synthetic datasets. The theoretical facet is focused on modelling the initialisation stage of BioHEL to determine the limitations of the system on certain domains. Afterwards, based on the conclusions from the analysis, we develop an automated parameter setting approach that determines the characteristics of the problem and sets the coverage breakpoint based on this knowledge. Moreover, to improve the execution time of the algorithm a parallel evaluation process using GPGPUs is developed. Finally, we introduce post-processing routines that can help reduce the amount of rules and number of attributes in the rules, to increase the generality of the solutions generated using the IRL approach.

1.3 Structure of the Thesis

This thesis is organised into ten chapters: the introduction, two preliminary chapters that present the context and background material on which the thesis builds upon, six content chapters that contains the contributions of this work and one final chapter that underlines the conclusions and further work.

Chapter 2 presents the general background work of this thesis. This includes the definitions and taxonomy for machine learning and data mining. This chapter also introduces EL, the representations commonly used and the state-of-the-art techniques used in this field to handle large scale domains. Moreover, the two systems analysed extensively in this thesis, GAssist and BioHEL, are explained in detail.

Chapter 3 presents the common methodological aspects shared among the different chapters. These are the statistical tests used, the datasets (which can be divided in to real-world and synthetic ones) and the basic configuration for the BioHEL system, which remains unchanged unless stated otherwise.

Chapter 4 **Pittsburgh vs. IRL: Identifying future challenges** presents a thorough comparison between the GAssist and the BioHEL system, it which the common mechanisms among the two are analysed to determine if they can be set up in the same way, or they depend on the learning paradigm used. Moreover, this chapter introduces a standard configuration that can be used for both algorithms depending on the general characteristics of the problem (whether the problem can be categorised as big or small). However, in the case of BioHEL it is shown that better results can be achieved if the coverage breakpoint parameter is set up according to the specific characteristics of the problem. Moreover, other areas of possible improvement in BioHEL are the execution time and the length of the rule sets generated using IRL.

Chapter 5 **Parameter impact in BioHEL's fitness function** presents an exhaustive experimentation over the coverage breakpoint parameter using synthetic boolean problems such as the k -Disjunctive Normal Form formulas. These formulas are introduced in this chapter as a way to characterise the structure of boolean problems based on the number of relevant attributes (k) and the number of terms in the problem (r) (rules the system needs to learn). In this chapter empirical conclusions are determined about how the coverage breakpoint parameter should be set up according to the characteristics of the problem. The conclusions state that if the problem has k relevant attributes in each term, the optimal coverage breakpoint for the system should be 2^{-k} .

Chapter 6 **Modelling the initialisation stage of BioHEL** derives theoretical models of the success of the initialisation stage to understand better how the system should be configured. In this chapter the schema bound (probability of obtaining good rules in an initial population) and covering bound (probability of covering the whole search space) are calculated based on system parameters and characteristics of the problems such as number of relevant attributes (k) and number of terms (r).

Chapter 7 **Automatic theory-based adjustment of the coverage breakpoint** introduces an automated parameter setting approach that determines the characteristics of the problem based on the models generated in Chapters 5 and 6 and observable characteristics in sample individuals. The models are used to determine both the number of relevant attributes in the terms (k) and the number of terms (r). Afterwards, based on this knowledge and the conclusions of Chapter 5, this method sets up the adequate coverage breakpoint for the problem.

Chapter 8 **Fast evaluation process using GPGPUs** tackles a different problem than previous chapters: the efficiency of the evaluation process. This chapter introduces a methodology to boost the evaluation process of BioHEL using the massively parallel computational capacity of General Purpose Graphic Processor Units (GPGPUs). Two different strategies with different degrees of parallelism are proposed and evaluated. The assessments of the efficiency improve-

ment are done over the evaluation process independently and over the whole learning process. Additionally, models that explain the results obtained are presented.

Chapter 9 **Post-processing the final rule sets** presents three post-processing operators that try to improve the generality of the rules. Since the final solutions generated using IRL might lack global supervision, it is probable that the solutions have more rules than necessary and these rules have more attributes expressed than they should. This chapter introduces three operators: *rule cleaning*, *rule pruning* and *rule swapping*. While the first two act on individual rules to decrease the number of attributes expressed, the last one swaps the order of the rules to find a new order in which unnecessary or redundant rules can be erased.

Chapter 10 contains the summary of the thesis, final remarks and further work.

1.4 Published Work

The work presented in this thesis have been published in three journal papers (two still under review process) and three conference papers. Here we present the list of publications, their relation with the chapters in this thesis and the associated prizes or awards received.

Journal papers

- Franco, M. A., Krasnogor, N., and Bacardit, J. (2012a). Analysing BioHEL using challenging boolean functions. *Evolutionary Intelligence*, 5:87–102. 10.1007/s12065-012-0080-9
Chapter 5: **Parameter impact in BioHEL's fitness function**
- Franco, M. A., Krasnogor, N., and Bacardit, J. (2013b). GAssist vs. BioHEL: critical assessment of two paradigms of genetics-based machine learning. *Soft Computing*, pages 1–29. Available online: <http://dx.doi.org/10.1007/s00500-013-1016-8>
Chapter 4: **Pittsburgh vs. IRL: Identifying future challenges**

Journal papers under review process

- Franco, M. A., Krasnogor, N., and Bacardit, J. (2013a). Auto-tuning a rule-based machine learning algorithm via problem structure identification. *Journal of Machine Learning Research*. Submitted for review on January 2013
Chapter 7: **Automatic theory-based adjustment of the coverage breakpoint**

Conference papers

- Franco, M. A., Krasnogor, N., and Bacardit, J. (2012b). Post-processing operators for decision lists. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference, GECCO '12*, pages 847–854, New York, NY, USA. ACM Press
Chapter 9: **Post-processing the final rule sets**
- Franco, M. A., Krasnogor, N., and Bacardit, J. (2011). Modelling the initialisation stage of the ALKR representation for discrete domains and GABIL encoding. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation, GECCO '11*, pages 1291–1298, New York, NY, USA. ACM Press
Chapter 6: **Modelling the initialisation stage of BioHEL**
2011 Best paper award for the GBML conference track

- Franco, M. A., Krasnogor, N., and Bacardit, J. (2010a). Speeding up the evaluation of evolutionary learning systems using GPGPUs. In *GECCO '10: Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 1039–1046, New York, NY, USA. ACM

Chapter 8: Fast evaluation process using GPGPUs
2010 Best paper award for the GBML conference track

CHAPTER 2

Background Work

This chapter focuses on describing the state-of-the-art in Machine Learning and Data Mining which is the broad context of this thesis. Additionally, the concepts of Evolutionary Computation and Genetic Algorithms are introduced. Afterwards, the field of Evolutionary Learning, the specific context of this thesis, is thoroughly described. The different problem domains over which Evolutionary Learning algorithms can be applied, rule-based knowledge representations and the efficiency enhancement techniques proposed within this field in the past years are explained in detail. Finally, this chapter explains in-depth the systems analysed in this thesis: BioHEL and GAssist.

2.1 Machine Learning and Data Mining

According to Mitchell [1997], Machine Learning (ML) corresponds to the research field that develops algorithms which improve their performance over a certain task with experience. However, there are many ways in which the concepts of *task*, *performance* and *experience* can be interpreted. For example, for a program that plays a particular board game, the task is to play the game, the performance measure is how good the program is at playing in a particular moment and the experience corresponds to all the games the program has played so far to reach its level of abstraction. In general ML algorithms are known from being able to obtain empirical data as inputs and learn or identify patterns within this data.

So far machines cannot be programmed to learn in the same way as humans do. However, there are many different tasks that machines can learn how to perform nowadays such as playing checkers, controlling robots, classifying and extracting knowledge from data, among others. Even though the research in the ML field exists since the 1950's, it was not until the 1980's considered an independent research field. Michalski et al. [1985] presents a compilation of the early work on this area.

Data mining consists in the process of automatically extracting knowledge or abstractions from a set of raw data [Witten and Frank, 2005]. Data mining can be considered as one of the usages that ML can have. While the terms ML and data mining are strongly related, the difference between the two is that data mining methods focus on extracting *unknown* knowledge from data, while the ML methods are more focused in generating better and more accurate learning algorithms, that might have a data mining purpose, but not necessarily.

There are many different ML paradigms and many different classification schemes [Langley, 1996]. The most popular classification schemes for ML are based on which information is available during the learning process and which knowledge representation is used to represent the final output. Depending on which information is available during the learning process, the learning can be:

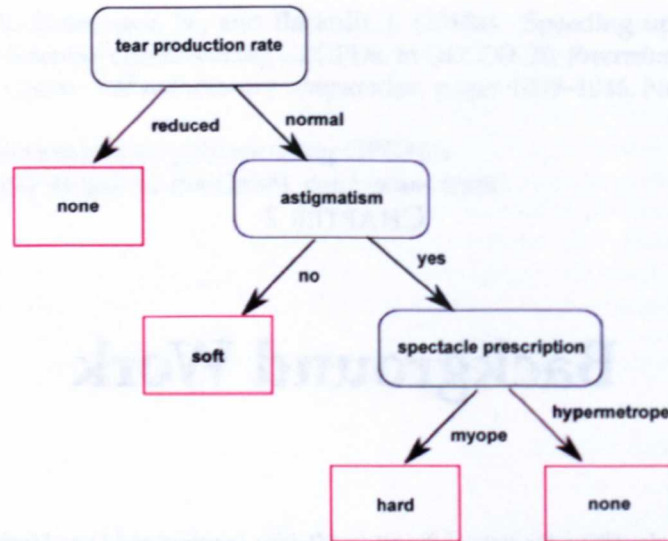


Figure 2.1: Example of decision tree for the contact lens prescription problem taken from [Witten and Frank, 2005]

Supervised when the inputs of the system have specific labels (the correct output for each case), which is determined previously by a human. In this type of learning there is direct feedback on how good is the underlined model so far. This type of learning is usually used to solve *classification problems* which consist in classifying or separating the instance space, based on a training instance set labelled with a finite number of classes. This thesis is mainly focused on this type of learning.

Unsupervised when the inputs do not have labels, but instead the system needs to find relations or similarities between the inputs. In this case there is no feedback on the performance of the model.

Semi-supervised is a middle point between the two previous types, in which some of the inputs are labelled and some of the inputs are not.

Reinforcement learning when the inputs do not have explicit labels, but instead the system receives an indirect feedback of how good or bad the prediction was (reward).

Furthermore, as it was mentioned before, the learning can be classified depending on the type of knowledge representation used to express the output [Witten and Frank, 2005]. In supervised learning (which is the focus of this thesis) the different knowledge representations are:

Decision trees. Consist in trees in which each node separates the instances according to an attribute and the leaves correspond to the final classification or label. An example of a decision tree can be found in Figure 2.1. Also decision trees can be used to predict numeric values instead of categories, by averaging the outcome of the instances that reach a particular leaf. This type of trees are called *regression trees*. One of the most representative algorithms that use decision trees is C4.5 [Quinlan, 1993].

Classification rules. Consist in rules that classify all the instances that match a condition C into a class A from a set of possible classes. The condition C corresponds to a boolean clause that holds for a subset of the input space. The class A can be either a label or an action to perform in the cases where the condition C holds. Therefore, the rules represent an implication "IF C THEN A ". The output of the systems that use rules usually consists of several classification rules that match different areas of the input space and can be applied in a hierarchical or in a non-hierarchical way. When the rules are applied in a hierarchical way the output is also referred as a *decision list* [Rivest, 1987], because the order of the rules

att1: tear production rate
att2: astigmatism
att3: spectacle prescription

- Rule 1:** $att1 = reduced \rightarrow none$
Rule 2: $att1 = normal \wedge att2 = no \rightarrow soft$
Rule 3: $att1 = normal \wedge att2 = yes \wedge att3 = myope \rightarrow hard$
Rule 4: $att1 = normal \wedge att2 = yes \wedge att3 = hypermetrope \rightarrow none$

Figure 2.2: Example of rule set for the contact lens prescription problem

determines a precedence in the activation. This means that to classify an instance, the class will be determined by the first rule in the rule set that matches the instance. An example of rule set is given in Figure 2.2. Examples of modern systems that use classification rules are JRip [Cohen, 1995] and PART [Frank and Witten, 1998]. Classification rules, and in particular decision lists, are the type of knowledge representation used by the systems analysed in this thesis. Therefore, more examples of systems that use classification rules from an EL perspective will be provided further in this chapter.

Linear models. Consist in a mathematical model that, based on the input, is capable of separating or classifying the solution space. In its simplest form consists of weighted sum of the attribute values. The simplest method to generate linear models is a linear regression. An example of a classification using a linear model can be found in Figure 2.3a. Among the approaches that generate linear models we can find Neural Networks [McCulloch and Pitts, 1943] and Support Vector Machines (SVM) [Vapnik, 1995]. Neural networks consist in a series of *perceptrons* interconnected. A perceptron is the minimal form of neural network which receives a series of inputs and produces an output that consists of a weighted sum of the inputs. The perceptrons connected together determine a final output as shown in Figure 2.3b. The weights are adjusted during the training process based on the prediction error. On the other hand, SVM constructs a hyper-plane (or a series of hyper-planes) that separates the instances. The hyperplanes are constructed trying to maximise the distance to all the different points in space. Particularly, SVM can separate non-linear spaces by means of using kernel transformations.

Instance-based representations is a type of representation that stores the observed instances and, when trying to classify a new one, finds the instance(s) with more resemblance to the observed one. Systems that used instance-based representations do not generalise,

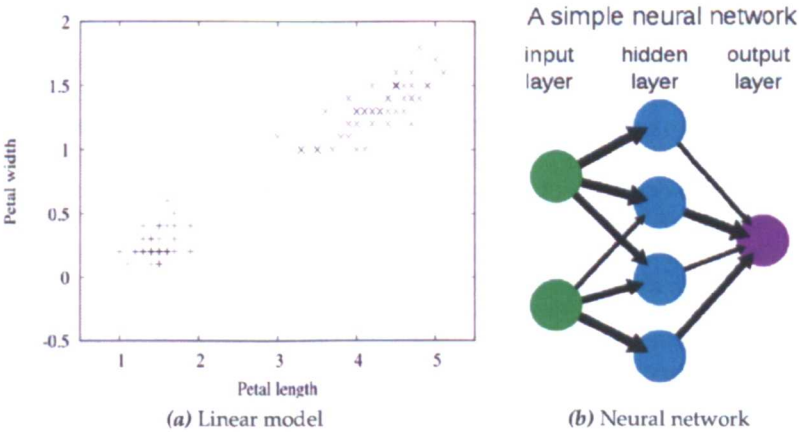


Figure 2.3: Example of a linear model that separates the data in two classes (left) and a neural network (right). Image taken from [Witten and Frank, 2005] and Wikipedia, respectively.

but instead predict using previous specific knowledge (instances stored in memory). One of the most popular algorithms that use this representation is the K-neighbours classifier (IBk) [Aha et al., 1991].

Bayesian networks is a probabilistic representation in which the system underlines a model that expresses the probability of a certain instance belonging to a certain class. This type of representation could also be classified as a linear model. Among the systems that used this type of representation we can find Naïve Bayes [John and Langley, 1995] and K2 [Cooper and Herskovits, 1992].

Ensembles is the combination of the prediction of several simple models [Rokach, 2010]. It can be considered a meta-representation as it works on top of the representations mentioned above. The combined prediction can be calculated using *bagging* (consensus prediction) where each one of the models have a vote (or a weighted vote) which contributes to the final prediction. At the end the class with more votes is selected as the winner. Other form of ensemble technique is *boosting*, in which the models are generated incrementally based on the instances that were poorly classified by the models already developed. This technique tries to generate complementary models. An important representative of an ensemble-based classifier is AdaBoost [Freund and Schapire, 1996].

2.1.1 Data mining for large scale datasets

When the datasets become very large, there are limitations on the application of ML techniques because of the memory requirements and the execution time. Particularly if the algorithms do not scale linearly according to the size of the dataset, trying to solve bigger and bigger datasets is soon unfeasible. In this section we provide a brief introduction of the techniques available for general data mining over large scale datasets. Even though this section does not mention particular details of the techniques, the ones are relevant to the work conducted in this thesis will be explained better from an EL perspective later in this chapter.

To reduce the complexity of the datasets it is possible to work with a subgroup of attributes or instances [Witten and Frank, 2005]. This does not only help decreasing the execution time, but also for some problems it can improve the quality of the built models. Removing attributes that are irrelevant and redundant from the data is called *feature selection*. This is a very important step when working with problems with large amounts of attributes and very low number of samples. Feature selection can be separated in two groups: feature ranking and subset selection [Guyon and Elisseeff, 2003]. Feature ranking ranks the available variables by a metric and eliminates or filters all those variables that achieve a score under a certain threshold. Subset selection techniques use optimisation algorithms to search the set of possible variables and find the optimal subset.

On the other hand, learning from a smaller sample of the training set is called *subsampling*. The subsampling techniques can be separated in two big groups: the ones applied before the learning process and the ones applied during the learning process. The subsampling can be done either in a supervised way, determining which instances are better representatives of the data or in a random way. The techniques that analyse and select the instances that are more important or relevant fall into the category of *instance selection* techniques. Also the subsampling can be performed during the learning process in a dynamic way. Windowing systems are examples of these techniques, which either partition or select instances from the training set dynamically. This means that the subset of the instances used change during the learning process.

Moreover, it is possible to reduce the execution time but adjusting parameters that regulate the complexity of the solutions obtained. The more complex the solution the more time it takes to generate it. An example of this is constraining the number of support vectors in SVM to improve the scalability over large datasets [Wang et al., 2012]. Many other examples of balancing the tradeoff between complexity and efficiency by means of parameters will be presented throughout this thesis.

Furthermore, parallelisation of the ML algorithms is possible. Some algorithms have a more parallel nature than others so the benefit from the parallelism depends greatly on the type of algorithm. Parallelism can be achieved via parallel programming tools, which allow distributing the work in different cores, or via high-level problem abstractions as the ones provided by the data-intensive computing frameworks. [Zaki and Ho, 2000] presents a compilation of parallel and distributed techniques to speed up data mining on large scale domains. Also other examples of data mining of large scale datasets have benefit from the usage of cloud computing (infrastructures that provide computational capacity services over the internet) [Grossman and Gu, 2008].

If the problem is the memory requirements, streaming can be used to provide the system with one instance at the time, without storing it in memory. However, this is only possible for algorithms that do not need to memorise the instances seen (e.g. Neural Networks). Also it is possible to perform data mining over data distributed in different physical places by means of data-intensive computing.

Examples which handle large scale domains are the application of data mining to web purposes, which usually involves large amounts of data [Liu, 2007]. For example, nowadays many websites, social-networks and search engines focus on providing information to the users according to their taste, which is defined by their previous actions [Chang, 2008; Shi et al., 2011]. Likewise, data mining with medical purposes, such as the determination of relationships among genes [Glaab et al., 2012; Urbanowicz and Moore, 2010] or the protein structure prediction [Bacardit et al., 2009b, 2012], involves really large amounts of data. Large scale data mining has also been applied in astronomy for image processing and object identification (e.g. quasar detection, galaxy classification) [Ball and Brunner, 2010], in healthcare for treatment effectiveness and healthcare management [Koh et al., 2011] and in fraud detection [Phua et al., 2010]. Weiss and Indurkha [1998] presents an interesting discussion of the revolution of big data and the challenges encountered by the data mining and ML fields.

2.2 Evolutionary Computation and Genetic Algorithms

Evolutionary Computation (EC) refers to the artificial intelligence field concerned with the application of Evolutionary Algorithms (EAs) to optimisation tasks. EAs are algorithms that apply metaphors taken from the natural evolution process. These algorithms try to simulate the natural selection first proposed by Darwin [1909], and the transmission of information from parents to offsprings according to the Mendel principles. Usually these algorithms develop a population of individuals (each of them a candidate solution for the problem) according to a pre-established objective. To do this they apply the Darwinian principles, such as the reproduction and survival of fittest and transmission of genetic information from parents to offsprings. According to [Freitas, 2002] the EC field is now divided in four branches: Genetic Algorithms, Evolutionary Strategies, Evolutionary Programming and Genetic Programming, being the first one the most relevant in the context of this thesis. The following paragraphs explain briefly each one of these paradigms.

Genetic Algorithms (GA) [Goldberg, 1989; Holland, 1975] Is the most widespread paradigm of evolutionary computation. This paradigm applies most of the Darwinian principles such as selection, crossover, mutation and inheritance. In a basic GA each one of the individuals in the population corresponds to a random solution for a given problem encoded as a *chromosome*. Through *crossover* and *mutation*, new solutions are generated based on good individuals of the population (recombining the information in their chromosomes), or applying random changes, respectively. According to the selection principles, the fittest individuals (the best solutions) stay in the population, while the worst ones should disappear to give room for better offsprings. The selection is based on the *fitness* of the individuals. GAs will be explained in greater detail in Section 2.3.1.2.

Evolutionary Strategies (ES) [Rechenberg, 1973] Is a paradigm of evolutionary computation that adopts the principles of mutation and adaptation. This paradigm characterises itself for the adaptation of the parameters of the system with the encoded solutions by including these parameters as features in the chromosome. This is also known as *self-adaptation* which will be explained in greater detail in Chapter 7.

Evolutionary Programming (EP) [Fogel, 1964] This paradigm of evolutionary computation is used to evolve finite state machines that act as predictors. It is very similar to Evolutionary Strategies, since the parameters of the program are evolved by means of EAs.

Genetic Programming (GP) [Cramer, 1985; Koza, 1992] Is the latest paradigm of evolutionary computation to appear in time. It consist in the usage of the genetic principles to evolve programs that solve a particular task. The difference between evolutionary programming and genetic programming is that the latter evolves the structure of the program itself.

Within the field of EC, this thesis is focused in the application of GAs for the discovery of classification rules. The following sections will explain in greater detail the field of EL, which is the specific ML field to which this thesis belongs to.

2.3 Evolutionary Learning

The Evolutionary Learning (EL) term refers broadly to the application of GAs [Goldberg, 1989; Holland, 1975] to ML tasks. EL has gained the attention of the research community for its ability to perform well in problem domains like classification tasks, reinforcement learning and function approximation [Bacardit et al., 2007a; Lanzi, 2008; Urbanowicz and Moore, 2009]. Moreover, there are a lot of successful applications of EL systems in robotics [Butz and Herbot, 2008; Hurst and Bull, 2006; Musilek et al., 2005; Stalph et al., 2009] biological data analysis [Bacardit et al., 2009b, 2012; Basset et al., 2011; Smith et al., 2010; Stout et al., 2008, 2009], medical data mining [Alayón et al., 2006; Glaab et al., 2012; Holmes, 1998; Llorca et al., 2007; Urbanowicz and Moore, 2010], optimisation [Franco et al., 2010b; Stone and Bull, 2008; Tabacman et al., 2008], among others.

The first method under the EL umbrella was first proposed by [Holland, 1975; Holland and Reitman, 1978] under the name of Classifier System (CS) as a model of an automata that could react to certain conditions in the environment. The way the automata or agent reacts to the environment is by means of classification rules which have the structure *IF condition THEN action*. All the rules learnt combined together form a behavioural model which is evolved by means of a GA. However, since then many different types of systems that fall on this category have been developed. In the following sections the main components of EL are underlined and the three learning paradigms of EL are explained.

2.3.1 Components of Evolutionary Learning

Although there are many different types of EL systems there are common characteristics among them that need to be highlighted. First, EL systems can be separated in two big groups: the ones that apply reinforcement learning and the ones that apply supervised learning. In the algorithms that apply reinforcement learning there are 4 basic components [Holmes et al., 2002; Lanzi, 2008]:

1. a population of classifiers that concentrates the knowledge of the system
2. a discovery mechanism which finds new solutions based on the previous ones
3. a credit assignment or reinforcement learning component

4. interaction between the classifier system and the environment (performance component)

In the algorithms that apply supervised learning the components 1 and 2 exist in the same way. However, the credit assignment and the interaction with the environment can be considered an unique process in which all the classifiers are evaluated against the whole training set and the system determines how well each one of them classify the set of instances. The following sections will explain each one of these components.

2.3.1.1 Population

EL systems usually handle a population of individuals which evolve through time following an optimisation process. The individuals usually consist of a rule or a rule set that can be encoded using many different knowledge representations as it will be explained later in Section 2.3.4. The representation consists of the *genotype* (the genes or encoding) and *phenotype* (its interpretation) [Kovacs, 2011]. In some representations these two concepts are the same while in some others the phenotype needs to be derived through a complex process.

When the individuals are rules, usually the whole population constitutes the solution of the problem, and the individuals collaborate among themselves to create this solution. In other systems, the individuals are rule sets. This mean, each individual constitutes an independent solution of the problem. They do not collaborate directly but the entire solutions are recombined to generate new ones.

2.3.1.2 The discovery component

The discovery component in EL is in general a GA [Goldberg, 1989; Holland, 1975]. As it was explained before, the individuals, whether they are an individual rule or a rule set, are encoded in a chromosome which has a measure of quality associated to it (*fitness*). In the original GAs this consists of a formula that evaluates the solutions. In EL the fitness of the individuals depends on applying the rules to particular examples or inputs. When using reinforcement learning the fitness tends to be related to the payoff (reward) or the accuracy at receiving a particular payoff. When following this approach a credit assignment mechanism is necessary to pass the environment's feedback to the classifiers as it is going to be explained in the next section. Moreover, in the supervised learning approaches the fitness is often based on the misclassifications over the whole training set.

The GA used by EL systems usually consists in the following steps or mechanisms:

Selection is the mechanism that determines which individuals are going to be evolved. This mechanism gives preference to the fittest individuals to pass their information to the next generations. This mechanism is stochastic with a probability that depends on the fitness of the individuals.

Crossover is the random recombination of the chromosome information of two parent individuals (determined by the selection mechanism) to generate two offsprings.

Mutation is the random alteration of the parent chromosomes, which is used to perform local search around the selected solutions.

Replacement is the deletion of the worst individuals from the population to make room for emerging offsprings, as the size of the population is finite. This mechanism is also stochastic and occurs on each individual with a probability inversely proportional to their fitness. The use of this mechanism should result in a global increase of the quality of the population.

These steps are repeated a fixed number of times or until the global quality of the population has reached a certain threshold. Moreover, since the selection and deletion mechanisms are stochastic with a probability that depends on the fitness of the individuals, the best individuals of the population are kept at all times, which is also known as *elitism*.

2.3.1.3 Credit assignment or reinforcement learning component

The credit assignment or reinforcement learning component allows the systems that interact with an environment to pass the feedback information back to the individuals [Holland, 1986a,b]. Depending on the input the automata receives from the environment, certain rules are applied. Based on the outcome of applying these rules, the fitness of the rules changes. This information feeds the GA to differentiate between “good” and “bad” individuals, while evolving the model that best fits the observable environment.

The first credit assignment method used in reinforcement learning was the “bucket brigade” algorithm [Holland, 1986a]. This algorithm distributes the feedback among all the classifiers that implied the chosen action. Afterwards, this credit assignment mechanism was extended to include temporal difference [Sutton, 1988]. Then, a modification of Watkins Q-Learning [Watkins, 1989] was proposed as the credit assignment mechanism of more complex systems.

In the case of supervised learning approaches there is no need for a sophisticated credit assignment mechanism. The “good” and “bad” rules or rule sets are determined based on the accuracy at classifying the available examples.

2.3.1.4 Interaction with environment

The interaction with the environment is the application of the generated rules to the problem at hand [Kovacs, 2011] to obtain a feedback on the classification quality from the environment. However, in some systems that apply supervised learning, the interaction with the environment does not exist explicitly, but only constitutes the calculation of prediction accuracy based on a training set. However, for systems that apply reinforcement learning the interaction is explicit, as the feedback depends on this interaction and the latter might even change the conditions of the environment.

2.3.2 Paradigms of Rule-based Evolutionary Learning

Depending on the type of learning and representations used different types of EL systems exist and they can be clustered in three main paradigms. The three different perspectives for learning rules in EL are: the Michigan approach [Holland, 1975], the Pittsburgh approach [Smith, 1980, 1983] and the IRL approach [Venturini, 1993]. The two first approaches were developed in parallel while the last one appeared later inspired in the previous ones. Moreover, the first paradigm usually applies reinforcement learning while the other two usually apply a supervised learning approach. However, hybrid systems can be also found in the literature. The following sections will explain the characteristics of these paradigms.

2.3.2.1 The Michigan Approach

The Michigan approach was first described by Holland [1975] under the name of *cognitive systems*. Each individual in the population represents one rule which has a condition *C*, an action *A* and an expected payoff. The population, as a whole, corresponds to the solution of the problem. The main goal is to develop a population that represents a payoff map, by means

of reinforcement learning. This means a model based on rules which express: if C happens and A is carried out the expected reward or payoff is P . This approach also has the ability to adapt itself to new examples that arrive to the training set and it is also referred to as “online learning”.

The main cycle of Michigan LCS goes as follows. When the system receives a training example, a match set $[M]$ is constructed with all the rules in the population that match the incoming example. If the match set is empty coverage occurs to generate random rules that cover the example. From all the actions that can be found in $[M]$, the payoff prediction is computed. The prediction is computed as the weighted fitness average of the payoff of the classifiers that imply this action. This means that the classifiers with higher fitness contribute more to the payoff prediction than the rest. Then an action is selected and the action set $[A]$ is constructed with all the classifiers in $[M]$ that implied this action. There are different methods to select the action: a) exploitation, which is the selection of the action with the higher expected payoff or b) exploration, in which a random action is selected to explore the solution space. Once the action is carried out, a reward is obtained from the environment and all the classifiers inside $[A]$ are updated. After this point, a GA is applied and two parents are selected to generate offsprings through mutation and crossover. Afterwards, the offsprings are introduced in the population and the cycle starts again. Figure 2.4 shows an example of the functioning of this paradigm taken from [Bacardit, 2004].

The first implementations of Michigan LCS used the *payoff* or *strength* as a measure of *goodness* of the classifier. One of the main representatives of Michigan LCS that used the payoff is the Zeroth-level Classifier System (ZCS) [Wilson, 1994]. Since these implementations used the payoff of the rules, they constructed a *best action map* instead of a complete landscape of payoff. This means that the rules with the highest payoff take over the system, because they are preferred without considering how accurate they are and the system does not know actually in which cases the payoff might be low. As a consequence, if the problems have a deceptive payoff the system will not be able to learn the most accurate solution to the problem. This changed with the introduction of the *accuracy* as a measure of *goodness* in the XCS system [Wilson, 1995], which helped the system not only to improve its performance but also its generalisation capabilities. Additionally, XCS applies the GA on the action set only, instead of applying it over the whole population. This comes from the idea that different niches of the environment can have different payoffs. If the GA is then applied over the whole population, the niche with the highest payoff will have more classifier resources explaining its behaviour than the rest of the niches.

While most of the systems that follow the Michigan approach use reinforcement learning, there are hybrid systems that apply supervised learning such as UCS [Bernadó-Mansilla and Garrell, 2003; Orriols-Puig and Bernadó-Mansilla, 2008b]. This system is based in XCS but instead of generating a *complete action map*, it generates a *best action map*, similar to the one generated in *strength-based* learning classifier systems. To do this they used a supervised learning scheme instead of reinforcement learning. The difference is that the supervised learning scheme provides the correct class, while the reinforcement learning scheme only provides a reward. The correct class determines which classifiers managed to classify correctly. Then UCS instead of generating an *action set* it generates a *correct set* and an *incorrect set* depending on the correct class. The GA is applied only on the correct set, since this represents the niches in UCS.

2.3.2.2 The Pittsburgh Approach

The Pittsburgh approach follows a more canonical supervised learning approach where a complete training set is available and rule sets are learnt in a batch fashion. This approach was initiated at the University of Pittsburgh by De Jong [1988] and Smith [1980, 1983]. In this approach each individual is a rule set (usually of variable length) that constitutes a complete solution of the problem. The performance of the rule sets is generally based on the misclassifications

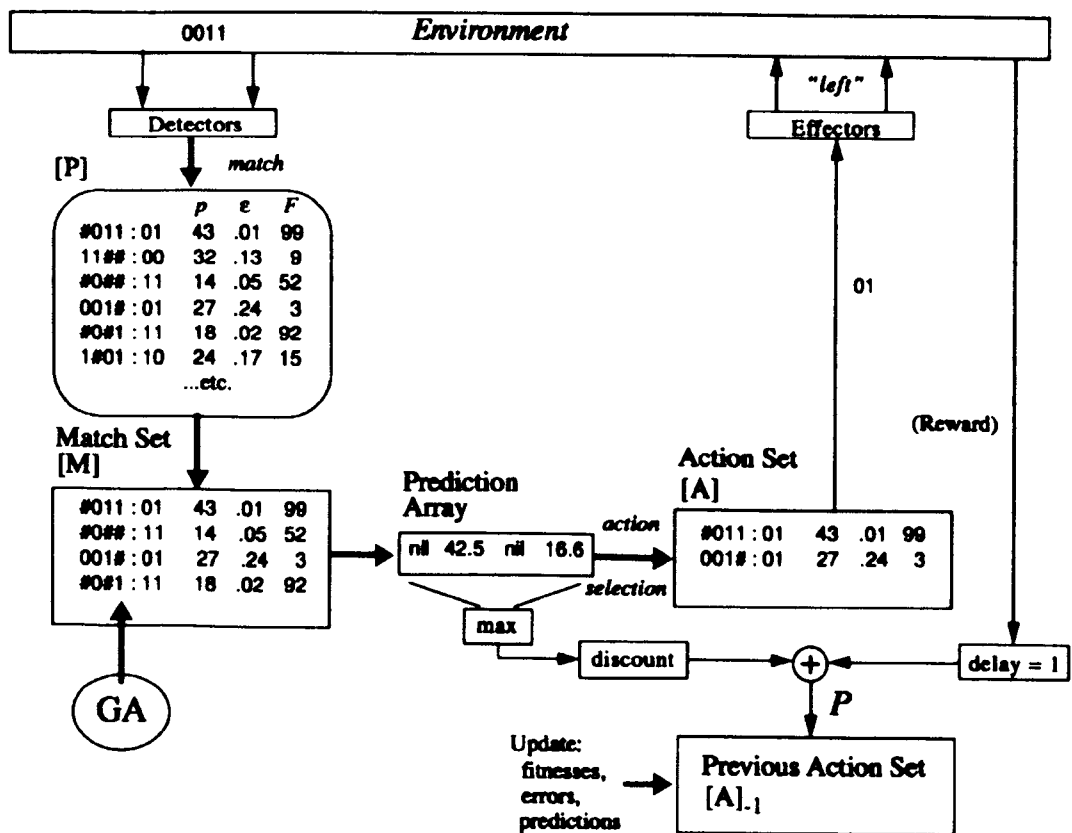


Figure 2.4: Flow of Michigan XCS taken from [Bacardit, 2004]

and the length, and it is calculated through supervised learning. This means that each one of the rule sets is tested against the whole training set in each GA iteration, which involves a higher computational cost than in the Michigan approach. While the latter rewards the rules that are responsible for the last action, the Pittsburgh approach evaluates how well each one of the solutions performs as a whole.

Since each individual constitutes a single solution (rule set), the way the Pittsburgh approach works is very similar to a standard GA, as shown in Algorithm 2.3.1. As other EL systems a GA is used to generate new solutions based on the existent ones. First, all the solutions are evaluated. Afterwards, using a selection mechanism two parent solutions are recombined, mutated and reinserted in the population with a certain probability, taking the place of the individuals with lower fitness. This process continues until a stop criteria is met, which is usually reaching a fixed number of iterations. At the end of the execution, the best individual in the population is considered the solution for the problem.

Algorithm 2.3.1: PITTSBURGHMAINLOOP(k)

```

pop ← INITIALISEPOP()
EVALUATE(pop)
stop ← TRUE
while ¬stop {
    pop2 ← SELECT(pop)
    RECOMBINE(pop2)
    MUTATE(pop2)
    // Inserts the new individuals in the population
    // considering the elitism
    MERGE(pop, pop2)
    EVALUATE(pop)
    CHECKSTOPCONDITION(stop)
}

```

Important representatives of this approach are GABIL [Jong and Spears, 1991], GIL [Janikow, 1993], GALE [Llorà and Garrell, 2001] and GAssist [Bacardit, 2004].

2.3.2.3 Iterative Rule Learning Approach

The Iterative Rule Learning (IRL) approach, consists in generating iteratively a set of rules that constitutes a complete solution of the problem. This approach was first used by Venturini [1993] in the SIA system. Similarly to the Michigan approach, each individual is one rule. But instead of having rules that cooperate to construct a solution, the solution is learnt incrementally, learning one rule at the time using consecutive GAs. Similar to the Pittsburgh approach the rules learnt correspond to the best individual of the population in a particular iteration. Furthermore, the fitness of the individuals is determined through a supervised learning approach, comparing the accuracy of the rules against the whole training set.

Algorithm 2.4.2 presents a pseudocode for the main loop of the IRL approach. The theory set (set of rules) starts empty and the rules are added to this set after being learnt independently by consecutive GAs. After each rule is learnt the examples covered by the new rule are erased from the training set and the execution continues until a stop condition is reached. In general this approach learns each one of the problem niches independently and ensembles the solution at the end constructing a hierarchical decision list.

Some authors consider this approach a variant of Pittsburgh, while others consider it a middle point between Michigan and Pittsburgh [Urbanowicz and Moore, 2009]. On one hand each individual is a rule like in the Michigan approach, but the process of learning rules in each

iteration correspond to the one in the Pittsburgh approach. Moreover, this approach is quite widespread in mainstream ML [Furnkranz, 1999].

Important representatives of this approach are HIDER [Aguilar Ruiz et al., 2003], NAX [Florà et al., 2009] and BioHEL [Bacardit et al., 2009a], which is the subject of study in this thesis.

Algorithm 2.3.2: ITERATIVERULELEARNING(*Examples*)

```

Theory ← ∅
while Example ≠ ∅
do
    Rule ← FindBestRule(Examples)
    Covered ← Cover(Rule, Examples)
    if RuleStoppingCriterion(Rule, Theory, Examples)
    then exit
    Examples ← Examples - Covered
    Theory ← Theory ∪ Rule
return (Theory)

```

2.3.2.4 On the nomenclature of Evolutionary Learning paradigms

As we have observed in the previous sections the three main paradigms of EL have many characteristics in common. However, the two first paradigms Michigan and Pittsburgh, belong to a subgroup of EL known as Learning Classifier Systems (LCS). While IRL has many similarities with the Pittsburgh and the Michigan approaches in my perspective it cannot be classified as LCS. This is because the set of rules that works as a classifier is generated sequentially while in the other approaches the classifier exists from the beginning and it is improved through time. At the same time EL systems are also referred to in the literature by many authors as Genetics-based Machine Learning (GBML) systems [Kovács, 2011].

As it was explained before, Pittsburgh LCS and IRL use GAs to perform batch learning using a supervised learning approach, while Michigan applies incremental learning often following a reinforcement learning approach.¹ While this thesis is mainly focused on the IRL approach, as part of the broader context of this thesis there is the Pittsburgh approach, Supervised Learning, Genetic Algorithms and Evolutionary Learning in general. To facilitate the reading Figure 2.5 presents a diagram that classifies all the techniques and paradigms presented so far and the relations among them.

Having explained the research fields that are related to this thesis, the rest of the chapter will be dedicated to explain the problem domains EL systems can tackle and some interesting remarks about knowledge representation and enhancement techniques, relevant to the work in this doctoral program.

2.3.3 Problem domains

EL systems have been applied widely in plenty of prediction problems. These prediction problems can be separated in three big groups: classification problems, reinforcement learning problems and function approximation problems [Butz, 2007].

¹ There are some examples in the literature in which Michigan applies incremental supervised learning as explained in Section 2.3.2.1

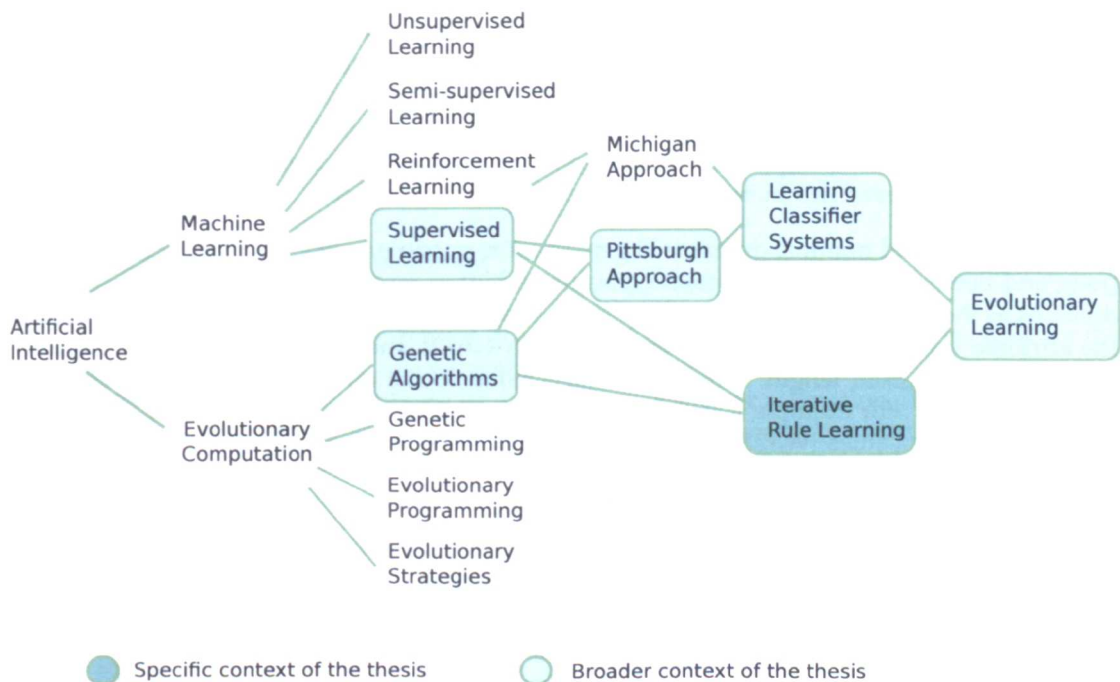


Figure 2.5: Diagram of the fields of research presented in this thesis. Dark blue represents the specific context of this thesis while light blue represents the broader context of the thesis

The *classification problem* is typically defined as a problem where the system has to identify areas in the problem space and label them with a certain class. A common example of this type of problem is to classify handwritten patterns by their meaning (letter or number expressed). Examples of the application of EL systems in this domain are [Bacardit and Butz, 2007; Bernadó-Mansilla and Garrell, 2003; Bernadó-Mansilla et al., 2006]. Moreover, the two systems analysed in this thesis have been tested before in this particular problem domain against other well-known ML techniques in [Bacardit, 2004; Bacardit et al., 2009a]. The *reinforcement learning problems* is a type of problem where the system is expected to develop a behavioural model based on rewards obtained from the environment. An example of this type of problem is finding the shortest path to the food in a maze. Examples of the usage of EL systems in this area are [Bull, 2001; Butz et al., 2005a]. The *function approximation problems* consist in fitting a continuous function. EL systems are capable of doing this by evolving a piecewise linear approximation of the function. The most important representative of EL applied to this type of problems is the XCSF system [Wilson, 2002]. An example of the application of XCSF to robotics using function approximation can be found in [Butz et al., 2009].

2.3.4 Knowledge representation

In the EL field a lot of effort has been focused in developing representations that are, not only more adequate for the problems we want to solve, but more efficient in terms of generality, expression power and performance. Even though, EL uses rules of the type *condition* \rightarrow *action*, both the way of representing the antecedent and the precedent are very flexible.

2.3.4.1 Representations for the condition

The representations for the condition part of the rules mostly depend on the type of problem at hand. For binary inputs we have the *ternary representation* $\{0,1,\#\}$ [Holland and Reitman, 1978], which consists of three possible symbols per problem attribute: 1 and 0 which force the

existence or non-existence of a particular attribute for the predicate to hold, and # (*don't care*) which represents that a particular attribute is irrelevant. For example the string 1#0# represents the following condition: *Att1* is *true* and *Att3* is *false*. Even though this representation is not very complex, it has been successfully used in [Goldberg, 1989; Lanzi, 2008; Wilson, 1995]. This representation can be extended to nominal domains by having one symbol per possible value per attribute. However, this increases dramatically the search space.

Also for binary (and nominal domains in general) it is possible to use the *GABIL* representation [Jong and Spears, 1991]. This representation has a higher representativity power in nominal domains as the predicate for each attribute is a disjunctive formula with all the possible values the attribute can take for the predicate to hold. In *GABIL* the attributes are represented by binary strings of fixed length. The length correspond to the number of possible values the feature can have. For example if the attribute *F1* may have the values (A,B,C), *F2* the values (O,P), and *F3* the values (W,Z,X,Y), a possible condition string for each one of the attributes would look like:

F1	F2	F3
100	01	1101

Each attribute is read as a disjunctive clause between all the values that have their bit on. For example, this condition can be interpreted as *if F1 is A and F2 is P and F3 is W or Z or Y*.

For real-valued domains several encodings have been proposed. The simplest and most widely used is the *interval-based knowledge representation*. In this representation a rule encodes an hyper-rectangle, defining for each attribute in the problem a lower and upper bound which defines the interval. Each interval corresponds to one side of the hyper-rectangle. Then all points that fall within the hyper-rectangle are matched by the rule. To represent the hyper-rectangles and determine the boundaries of the intervals during the learning process several representations have been proposed. First, we have the Centre-spread representation proposed by Wilson [2000] which consists in defining the interval as a centre c and a spread s . Then the lower and upper bounds are calculated as $(lb, ub) = (c - s, c + s)$. Another way of defining the dynamic intervals is the Min-max representation [Wilson, 2001b], in which each interval is explicitly represented by a lower and upper bound. Stone and Bull [2003] presented a throughout comparison of these two representations, which are the most wide-spread ones. Other representation for dynamic intervals is the Min-percentage representation [Dam et al., 2005a] which stores the lower bound and the percentage of the interval which corresponds to the distance between the lower bound and the upper bound. Also, other representations such as Adaptive Discretisation Intervals (ADI) [Bacardit and Garrell, 2003b], use pseudo-fixed intervals which remain constant for all the rules generated, but they can be adapted dynamically using multiple discretisation algorithms.

To use the generalisation capabilities of the systems, some EL implementations, such as NAX [Llorà et al., 2009], have a special encoding to represent the *don't care* interval. In this system the irrelevant interval is represented by a condition where the lower bound is larger than the upper bound.

More complex representations for real-valued attributes include ellipsoids [Butz, 2005], hyper-ellipsoids [Butz et al., 2006a], convex-hulls [Lanzi and Wilson, 2006] and neural networks [Bull, 2002; Bull and O'Hara, 2002; Howard et al., 2012]. However, our explanations are limited to the interval-based representations which are the ones used in this thesis.

Moreover, there is work in developing representations that use a semantical approach such as first order logic [Mellor, 2005], messy conditions [Lanzi and Perrucci, 1999a], fuzzy conditions [Casillas et al., 2007; Orriols-Puig et al., 2009] and S-expressions [Browne and Ioannides, 2007; Lanzi and Perrucci, 1999b]. This very particular representations are mainly developed to adapt themselves better to the characteristics of the problem and improve the generalisation capabilities of the systems.

However, one known drawback of using complex representations is that the size of the rule sets tends to increase. The way to tackle this problem is by developing post-processing routines that remove rules that are redundant or have a high overlapping with the rest of the rules [Butz et al., 2008b; Dixon et al., 2003; Fu and Davis, 2002; Wilson, 2001a]. These algorithms will be explained in greater detail in Chapter 9 where a new rule set compaction approach is introduced.

2.3.4.2 Representations for the actions

The representations used for the actions in a classifier are not as varied as the ones used for the condition. Usually the actions are represented by a set of symbols. However, there is recent work in which the action is replaced by a computed action or a function. For example Tran et al. [2007], implemented actions which are computed as a linear approximation using weights. Later Lanzi et al. [2007] implemented a similar idea, but learning the weights using supervised learning.

Also some approaches designed for function approximation such as XCSF [Wilson, 2002] do not have an action, but only use a computed prediction calculated using weights that change using reinforcement learning.

2.3.5 Efficiency enhancement methods

Given the need to tackle larger datasets each time, many efficiency enhancement techniques have been proposed in the field of EL [Bacardit and Llorà, 2012]. As it was explained before, running the GAs using training examples instead of a mathematical function to calculate the fitness of the individuals involves a huge computational cost that needs to be taken care of to maintain the scalability of the algorithms. These efficiency enhancement methods can be characterised into four categories: software-based solutions, parallel learning paradigms, hardware-based solutions, and data-intensive computing.

2.3.5.1 Software-based solutions

Software-based solutions are the ones in which particular algorithms or mechanisms have been developed to improve the performance of the systems without relying on specific hardware. Among these type of solutions we can find *windowing mechanisms* whose goal is to perform the match process in a more efficient way. These techniques consist in the selection of a representative subset of the training set to perform the fitness calculations. There are many ways in which this can be achieved. For example it is possible to select a fixed subset of instances of the training set and use this subset along the learning process. This is also known as *prototype selection* [Derrac et al., 2010; García et al., 2012]. Also the subset of instances can be selected dynamically by either selecting it randomly or selecting it based on observations over the instances like “misclassification ratio” or “age” (how long it is been since the instance was used for the last time). Moreover, smart stratification processes can be found in the literature such as the ILAS windowing scheme [Bacardit et al., 2004], the Random/Dynamic Subset Selection (RSS-DSS) [Song et al., 2005] and the Training Data Rotation [Ishibuchi et al., 2012]. The ILAS windowing scheme consists in separating the training set into non-overlapping strata and using a different strata in each GA iteration following a round robin policy. Since this technique is used by both systems analysed in this thesis it will be explained in greater detail in Section 2.4.1.3. On the other hand, the RSS-DSS approach performs two sampling process. First a Random Subset Selection is performed in a similar way as in the ILAS, creating non-overlapped blocks. Afterwards a Dynamic Subset Selection is performed within each block generated with RSS based on the age of the instances and their misclassifications. Finally, the Training Data Rotation consists of a windowing technique applicable to island model parallel EL algorithms

(See Section 2.3.5.2), in which the training set is distributed among the different islands and then rotated periodically among them.

Other software-based solutions are based on changing the representation of either the classifiers or the instances to perform the match process more efficiently. For example in [Giraldez et al., 2005] the instances were reorganised and stored in a tree structure that makes it very efficient to determine which instances match a particular classifier. Regarding the classifiers representation, some approaches [Butz et al., 2008a] reorder the attributes in a rule so the more specific ones appear first, and hence, decrease the time it takes to determine an example does not match a rule. Moreover, other representations such as the Attribute List Knowledge Representation [Bacardit and Krasnogor, 2009a] express only the relevant attributes in a list, avoiding the match operations of the irrelevant attributes. Since this representation is used in the systems analysed in this thesis it will be explained in greater detail in Section 2.4.2.1.

Furthermore, there are implementations of EL systems that use search mechanisms that try to explore the search space more efficiently. This is the case of the usage of Estimation of Distribution Algorithm (EDA) [Larraña and Lozano, 2002] and Memetic Algorithms [Krasnogor and Smith, 2005]. EDAs are optimisation methods that build a probabilistic model expressing candidate solutions in an incremental manner. Examples of the application of EDAs to EL are the XCS/ECGA and XCS/BOA [Butz et al., 2005b, 2006b], which apply the Extended Compact Genetic Algorithm (ECGA) [Harik, 1999] and the Bayesian Optimisation Algorithm [Pelikan et al., 1999], respectively. On the other hand, Memetic Algorithms are population-based algorithms with local refinement mechanisms over the rules that have a particular goal and do not act in a stochastic manner. Examples of the application of Memetic Algorithms in EL is the MPLCS system [Bacardit and Krasnogor, 2009b], which is a Pittsburgh LCS with four extra operators that try to modify the rules or the rule sets in an intelligent manner trying to minimise the error.

Moreover, another way to speed up the evaluation process is to substitute it with a fitness approximation or *fitness surrogates* [Jin, 2005; Llorà et al., 2007; Orriols-Puig et al., 2008c]. The fitness surrogates are usually based on a model of the problem previously determined using EDAs.

2.3.5.2 Parallel learning paradigms

Some EL systems have been designed and developed with a parallel nature in order to distribute the load of the evaluation process in a transparent way. An example of this is the GALE system [Llorà and Garrell, 2000; Llorà and Garrell, 2001] which combines the concepts of cellular automata and a supervised learning system. This system consists of a grid, in which each cell contains either one or zero individuals, and each individual is a complete solution of the problem. Each cell performs an evolutionary learning process in parallel over each individual, and the communication between cells is restricted to the neighbours and it is only performed during the recombination process.

Moreover, the main paradigms for GA parallelisation can also be applied to EL systems. The different parallelisation paradigms for GAs [Cantú-Paz, 1998] are:

Single population coarse-grained (Master-slave models). In these algorithms there is a single population and only the fitness function is parallelised among different processors. After the calculations are performed the information is returned to the master node to continue with the GA.

Single population fine-grained. In these algorithms there is a single population as well, but the individuals are organised in a spatial way, so the recombination operators are restricted to neighbours, limiting in that way the necessary communications.

Multi population coarse-grained (Island models): In these algorithms there are several populations which evolve in parallel, and exchange individuals or information occasionally through a “migration” operator.

Examples of EL systems that use classic parallelisation paradigms and/or Message Passing Interface (MPI; a communications protocol independent from the programming language used to program parallel computers [Gropp et al., 1996]) are [Bull et al., 2007; Dam et al., 2005b; Galizia et al., 2010; Llorà et al., 2009, 2007]. Probably the most interesting and relevant example to this thesis is the NAX system [Llorà et al., 2009] which applies the master-slave model to IRL and runs in parallel the same algorithm (using the same seed) over different parts of the population. Also [Cantú-Paz, 2000] developed a formal theory to set up parameters in parallel GAs, such as the size and the number of populations (in case of multi population algorithms).

Moreover, ensemble techniques, although they were not created with a parallelisation purpose, can help handling very large domains by generating independent models over different parts of the data and then merge them to create a single prediction model [Bauer and Kohavi, 1999; Breiman, 1999; Bull et al., 2007; Merz, 1999].

2.3.5.3 Hardware-based solutions

Hardware-based solutions correspond to the solutions that depend on a specific hardware to run. Examples of this are the usage of General Purpose Graphic Processor Units (GPGPUs) to parallelise the whole or parts of the GA. GPGPUs are graphics hardware initially designed for image rendering which can perform high performance calculations relying on its massively parallel architecture. The most widespread frameworks to program GPGPUs are NVIDIA’s Computer Unified Device Architecture (CUDA) [NVIDIA, 2008] and its open source alternative OpenCL [Scarpino, 2011].

There are many different examples of the implementation of GAs in GPGPUs. For instance, Maitre et al. [2009] presented an implementation of a GA which performs in parallel (following a master-slave model) the evaluation function of all the individuals using GPGPUs. Also, [Pospichal et al., 2010] presented a GA that runs in GPGPUs following the island parallelisation model. Moreover, [Li et al., 2007; Yu et al., 2005] present fine-grained parallelisation approaches implemented in GPGPUs. In these approaches the whole GA is implemented within the device. Since the communication between the GPGPU nodes are expensive, these approaches restrict the communications to high speed communications (neighbour nodes). Regarding Michigan LCS there are two works in the literature in which they parallelise the construction of the prediction array [Lanzi and Loiacono, 2010; Loiacono, 2011]. The usage of GPGPUs in the literature will be explained more extensively in Chapter 8 in which a novel approach to speed up the evaluation process for supervised learning is introduced.

Also another example of the application of specialised hardware is the usage of vectorial processor units. A vector processor is a Central Processing Unit (CPU) which is capable of applying in parallel a set of instructions to a vector or an array of elements, following the Single Instruction Multiple Data (SIMD) paradigm. An example of the usage of vector operations is [Llorà and Sastry, 2006]. In this work they speed up the match of binary conditions by using compact bitwise encodings for the rules and the instances. Afterwards, only bitwise *and* and *equal* operations were needed to determine if the condition of the rule matched the instance. This not only helps to perform the match faster but also reduces the amount of memory necessary to encode the rules and individuals. This approach was also extended to work with real domains in [Llorà et al., 2009].

2.3.5.4 Data-intensive computing

Data-intensive computing are paradigms of parallelism which use data parallel techniques to process very large amounts of data. These techniques are usually applied to algorithms that are data-bounded instead of computationally-bounded. This means that the algorithm is bounded by the processing time of the I/O operations and the manipulation of the data. The main characteristics of data-intensive computing frameworks are:

- The abstraction of the operations in a model that is independent to the architecture used and it is expressed in high level data-oriented operations.
- The locality of the data. Since these approaches usually try to minimise the movement of the data, they tend to run where the data is stored.
- They are robust to faults or errors. This is done by having redundant copies of the data, storing the intermediate results and not only the final ones, and having a system capable to automatically detect errors or failures.
- They can virtually scale linearly to process any amount of data by adding more computing nodes.

There are different data-intensive frameworks: MapReduce [Dean and Ghemawat, 2008], and its open source implementation Hadoop [Lam, 2010], Meandre [Llorà et al., 2008] and the ECL programming language for the High Performance Computing Cluster (HPCC) [Yoo and Kaplan, 2009].

The MapReduce architecture consists of a problem abstraction to two functional programming primitives that are independent from the architecture used: *map* and *reduce*. Both of these functions are defined by the user. The *map* function is in charge of converting an input to an intermediate key-value pair. The *reduce* function is in charge of merging the possible values of a particular key into one value. The way this is parallelised is by transparently distributing the computation of both the map and reduce functions. However, the user of this architecture does not need to understand the way the parallelisation works because the functions are written in high-level language which is independent from the underneath architecture. An easy example of the application of this architecture is counting the number of occurrences of a certain word w in a text. In this case the map function is in charge of generating the pair $(w, 1)$ every time the word is found and the reduce function is in charge of counting how many pairs were generated for the word w .

Moreover, Meandre [Llorà et al., 2008] is a semantic enabled web-driven dataflow execution environment, which provides the tools for assembling and executing *data flows*. Data flows are software focused in processing data (accessing, transforming and analysing or visualising the data and/or results). Each flow has a high-level representation as a graph that shows executable components.

Furthermore, the ECL (Enterprise Control Language) [Yoo and Kaplan, 2009] is a data-centric programming language in which programs do not exist, but only data and the operations to perform on that data are declared. This programming language has primitives which have a direct correlation to the MapReduce ones: *project* and *rollup*. Moreover, it was extra primitives that allow to sort, distribute, parse, normalise the data, among others. These primitives run transparently in parallel in the HPCC (High-Performance Computing Cluster), also known as DAS (Data Analytics Supercomputer) [Middleton, 2011].

There are several examples of GAs implemented using data-intensive computing. [Verma et al., 2009] shows an example of a simple selecto-recombinative GA implemented in Hadoop. Later in time, also the Extended Compact Genetic Algorithm (ECGA) [Harik, 1999] was implemented over Hadoop in [Verma et al., 2010]. At the same time, implementations of the two previous algorithms in Meandre were also proposed [Llorà, 2009]. [Llorà et al., 2010] presents a more

extensive survey of the application of data-intensive computing to GAs.

2.4 GAssist and BioHEL

There are two systems which are analysed in this thesis, GAssist and BioHEL. GAssist [Bacardit, 2004] is a Pittsburgh LCS with several characteristics to improve the scalability and the quality of the learning for real-world data mining problems. Among these characteristics we have a representation based on the discretisation of the real domain into intervals (using the ADI representation), and a fitness function that improves the generalisation capabilities of the system. However, its performance is limited in certain scenarios, such as problems with high number of classes and large training set sizes [Bacardit and Krasnogor, 2006]. This led to the introduction of BioHEL [Bacardit et al., 2009a], a system that shares several characteristics with GAssist, but changes the main learning paradigm to IRL in order to cope with these large scale domains. Also BioHEL discarded the mechanisms which took most of the time in GAssist and changed its fitness function to introduce the concept of *coverage* as an extra indicator of the goodness of a rule. These changes helped the system to scale better in complex bioinformatic domains. These two systems have been successfully applied to a very broad range of problems [Bacardit and Butz, 2007; Bacardit and Krasnogor, 2008a, 2009b; Bacardit et al., 2009b; Orriols-Puig et al., 2008a; Smith et al., 2010; Stout et al., 2008, 2009]. In particular, BioHEL has obtained very important results when solving protein structure prediction problems and has won related merits, such as being placed among the best algorithms for this purpose at the Critical Assessment of Techniques for Protein Structure Prediction (CASP) in 2008 and 2010, and winning the bronze medal at the HUMIES Awards for Human-competitive results in 2007.

While the focus of this thesis is to propose improvements for the BioHEL system, a comparison between the two systems will be presented in Chapter 4 as the start point to determine possible areas of improvement in BioHEL. Therefore, the following sections present in greater detail the characteristics of these two systems in the order in which they were created.

2.4.1 GAssist

GAssist [Bacardit, 2004] is a Pittsburgh LCS which started as an implementation of GABIL [Jong and Spears, 1991], hence it still uses the semantics of its representation and crossover operator. In GAssist, the individuals consist in a set of rules that represent a complete (and variable-length) problem solution. Solutions evolve using supervised learning and a standard GA. Finally, at the end of the execution, the best individual in the population in terms of fitness is considered to be the final solution to the problem.

The following section will explain in greater detail the characteristics of this system including its representation, fitness function and other efficiency enhancement techniques.

2.4.1.1 Representation

The GAssist system is capable of handling discrete and continuous attributes, by using different representations for each case. To handle discrete attributes the system uses the GABIL representation [Jong and Spears, 1991] as explained in Section 2.3.4.

To initialise GABIL attributes the system relies on a parameter p which corresponds to the probability of setting the bits in a GABIL string to 1. This parameter controls the *specificity* or *coverage* of the initial rules. These are two concepts that are linked to the percentage of examples covered by a rule. However, the relationship between them is inverse: the more specific the rule is the less coverage it has.

Moreover, if the covering mechanism is used a rule will be created setting the bits corresponding to the values of a randomly sampled instance to 1. For example if the attribute F1 may have the values (A,B,C), F2 the values (O,P), and F3 the values (W,Z,X,Y) and the instance {C, P, W} is sampled, the generated rule through covering would have at least the following bits on:

F1	F2	F3
1	*1	1*

The rest of the positions marked with * are set to 1 with probability p , similarly to when the system does not use covering. Moreover, the match process of a GABIL condition is performed as shown in Algorithm 2.4.1.

Algorithm 2.4.1: GABILMATCH(*Instance*, *Rule*)

```

match ← TRUE
for att ∈ Instance.atts ∧ match
{ match ← match ∧ Rule[att.index][att.value]
return (match)

```

On the other hand, to handle continuous attributes GAssist uses the Adaptive Discretisation Intervals (ADI) rule representation [Bacardit and Garrell, 2003b]. This knowledge representation generates intervals based on the output of multiple discretisation algorithms. The learning process chooses the correct discretisation for each problem. Moreover, the final intervals evolved in ADI can be the result of the merge of some of the original intervals produced by the discretisation algorithms (micro-intervals). Semantically, the evolved intervals follow the GABIL representation.

Moreover, this system uses a default rule mechanism [Bacardit et al., 2007b], which consists of a rule that covers all the examples left in the training set. This mechanism was introduced to generate more compact rule sets, since the default class is not used in the evolved rules. Therefore, if the problem has n classes the system will only generate rules for $n - 1$ classes. The default class is defined by a user established policy that can have the following behaviours:

Majority: the class which has the largest amount of examples is considered the default class.

Minority: the class which has the minimum amount of examples is considered the default class.

Automatic: the system automatically determines the most convenient default class.

Disabled: no default class is used. The system is able to evolve rules for all the classes in the dataset.

2.4.1.2 Fitness function

The rule sets in GAssist, as in most Pittsburgh LCSs, have variable length. One of the problems of the systems with variable length individuals is that the individuals might start growing without control. This phenomenon is known as the *bloat effect*. GAssist uses a fitness function based on the Minimum Description Length (MDL) principle [Rissanen, 1978] and a rule deletion mechanism to cope with this issue [Bacardit and Garrell, 2003a]. This fitness function considers two important characteristics of the rules: the accuracy and complexity (generality). It has two terms as shown in Equation (2.4.1).

$$F(R) = TL(R) \times W + EL(R) \quad (2.4.1)$$

$TL(R)$ (theory length) corresponds to the complexity (or generality) of rule set R , $EL(R)$ (exceptions length) corresponds to the accuracy of rule set R and W is a weight that adjusts the relation between the previous terms.

In particular, the term $EL(R)$ corresponds to:

$$EL(R) = 105 - \frac{\text{correctlyClassified}(R) \cdot 100}{|T|} \quad (2.4.2)$$

where $|T|$ is the number of instances in the training set and $\text{correctlyClassified}(R)$ is the number of instances classified correctly by rule set R .² The more accurate the rule is the smaller the term $EL(R)$ would be.

On the other hand, the theory length term $TL(R)$ penalises rule sets that have a complex representation, and therefore are less general. Considering that there are NA attributes in the problem and the rule set is of size $|R|$, this term can be calculated as shown in Equation (2.4.3) as the sum of the theory length of each one of the attributes belonging to the classifiers of the rule set.³ The definition of the theory length of each attribute depends on the employed knowledge representation.

$$TL(R) = \sum_{i=1}^{|R|-1} \sum_{j=1}^{NA} TL_{ij} \quad (2.4.3)$$

When an attribute is continuous, GAssist uses the ADI representation. Therefore, it calculates the term TL_{ij} as follows:

$$TL_{ij} = I_{ij} + NI_{ij} \quad (2.4.4)$$

where I_{ij} is the number of micro-intervals in the attribute j of rule i , NI_{ij} corresponds to the number of simulated intervals⁴ in the same attribute.

On the other hand, if the attribute in the rule is discrete, the system uses the GABIL representation and calculates the theory length as follows:

$$TL_{ij} = v_{ij} + n_{ij} \quad (2.4.5)$$

where v_{ij} is the number of possible values and n_{ij} is the number of negative values in the attribute j of rule i .

Moreover, as it was explained before the MDL formula depends on a weight W , which is adjusted dynamically throughout time using a heuristic defined in [Bacardit and Garrell, 2003a]. This value starts at a constant value equal to:

$$W = \frac{\text{initialRatio} * EL}{(1 - \text{initialRatio}) * TL'} \quad (2.4.6)$$

²This function sums 105 instead of 100 to handle border cases.

³This function only sums the theory length of the first $|R| - 1$ rules, since the last one corresponds to the default rule.

⁴Contiguous bits that have the same value, either true or false in the ADI representation.

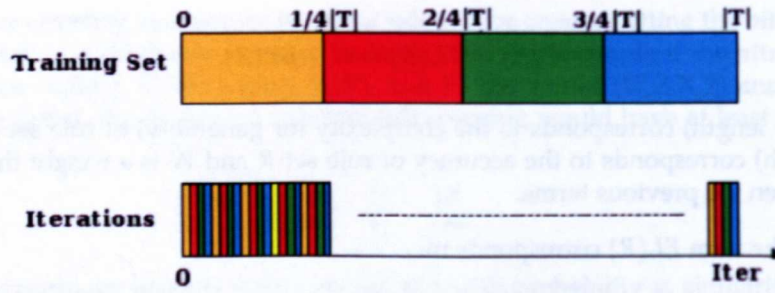


Figure 2.6: Example of the ILAS windowing partition and window usage.

$$TL' = TL \frac{NR}{NC} \quad (2.4.7)$$

where *initialRatio* is a parameter set by the user and the theory length TL is normalised as shown multiplying by the number of active rules NR and dividing by the number of classes NC . This is done to penalise rule sets with many dead rules (rules that don't match any examples). Afterwards, whenever the best individual of the population stays the same for a certain number of iterations the current weight is multiplied by a relaxation factor. Both the number of iterations and the relaxing factor are parameters set by the user.

2.4.1.3 ILAS Windowing Scheme

GAssist incorporates an efficiency enhancement mechanism called Incremental Learning with Alternative Strata (ILAS) [Bacardit et al., 2004], which is a windowing scheme that reduces the computational costs of handling large scale datasets. This technique separates the training set into strata, which have the same class distribution as the whole training set. Afterwards, each iteration of the GA uses a different strata for its fitness computations based on a simple round-robin policy as shown in Figure 2.6.

It has been shown empirically that this mechanism, not only improves considerably the execution time of the system, but also introduces more generalisation pressure over the system by preventing the over-fitting of the data [Bacardit et al., 2004]. However, there are limitations on its use since the correct functioning depends on each strata being a reliable representation of the whole training set. In order to calculate the probability of success of this approach it is necessary to calculate the probability of having an unrepresented niche as follows:

$$P(\text{unrepresented niche})_s = (1 - p)^{\frac{D}{s}} \quad (2.4.8)$$

where s is the number of strata, D is the size of the training set and p represents the probability that a random problem instance belongs to one niche. Based on the probability of having and unrepresented niche the probability of success can be calculated by Equation 2.4.9 where r is the number of niches in the problem.

$$P(\text{success})_s = (1 - P(\text{unrepresented niche})_s)^{rs} \quad (2.4.9)$$

2.4.1.4 Robust prediction using ensembles

Also ensemble techniques [Bacardit and Krasnogor, 2008a] were incorporated in GAssist to increase the robustness of the predictions. The ensemble techniques used are based on consensus prediction. This means that for a given training set several rule sets are generated using different seeds. Afterwards, the different rule sets vote to classify each example of the corresponding

test set. For each example the class that was predicted more frequently by the rule sets would be the final classification. If the generated models are diverse they might complement each other to improve the prediction. This ensemble scheme has shown to increase GAssist's performance across a broad range of problems [Bacardit and Krasnogor, 2008a; Stout et al., 2009]

Algorithm 2.4.2: BIOHELIRL(*Examples*)

```

Theory  $\leftarrow \emptyset$ 
noBestFound  $\leftarrow 0$ 
while Examples  $\neq \emptyset$ 
    {
    BestRule  $\leftarrow \text{NULL}$ 
    for  $i \leftarrow 0$  to numRepetitions
        {
        Rule  $\leftarrow \text{BESTRULEGA}(\text{Examples})$ 
        do {
            if BestRule < Rule
                then {
                    BestRule  $\leftarrow$  Rule
                    noBestFound  $\leftarrow 0$ 
                }
        }
    do {
        if COVERINGOK(BestRule)
            then {
                C  $\leftarrow \text{COVER}(\text{BestRule}, \text{Examples})$ 
                Examples  $\leftarrow$  Examples  $-$  C
                Theory  $\leftarrow$  Theory  $\cup$  BestRule
                noBestFound  $\leftarrow$  noBestFound + 1
            }
            else {
                if noBestFound = 3
                    then exit
            }
        }
    }
return (Theory)

```

2.4.2 BioHEL

The BioHEL (Bioinformatics-oriented Hierarchical Evolutionary Learning) system is an EL system proposed by Bacardit et al. [2009a] to handle large scale bioinformatic datasets [Bacardit et al., 2009b; Bassel et al., 2011; Stout et al., 2008, 2009]. As it was previously explained, this system inherits some components from GAssist such as the ILAS windowing scheme, the default rule mechanism and some parts of the fitness function, but instead of following the Pittsburgh approach to generate rule sets, it uses IRL. As it was explained before, this learning approach generates hierarchical rule sets with the particularity that rules are learnt sequentially, using a standard GA to learn each rule. Once a rule is learnt (which corresponds to the best individual in the GA population), it is added to the final rule set and all the examples covered by this rule are removed from the training set. This process is repeated iteratively until there are no more examples in the training set or the generated rules are not good enough. For instance if using default class, the stop criteria consists in not being able to find a rule better than the default rule three times in a row. The IRL paradigm used in BioHEL can be exemplified by the code in Figure 2.4.2. The following sections will explain in greater detail the characteristics of BioHEL that differ from GAssist.

2.4.2.1 Representation

BioHEL uses the Attribute List Knowledge Representation (ALKR) [Bacardit and Krasnogor, 2009a] to encode the individuals. ALKR is a meta-representation that is able to handle efficiently continuous and discrete attributes at the same time. It reduces the computational cost of matching the rules by representing in a list only the attributes that are relevant for the problem, hence avoiding spending extra computer cycles evaluating irrelevant attributes during a match

process. The relevant attributes may vary between the rules and they are determined independently for each rule during the learning process. In order to explore the space of attributes to identify the relevant ones, this representation has two extra operators, called *generalise* and *specialise*. These operators drop or add attributes from/into the rule, respectively. To handle discrete attributes within ALKR, BioHEL uses the GABIL representation [Jong and Spears, 1991] while a hyper-rectangle representation with explicit intervals [Wilson, 2001b] is used for continuous attributes.

Figure 2.7 shows an example of an individual using this representation. Each classifier condition is formed by five structures: a) an integer with the number of attributes represented b) a list of the identifiers of the represented attributes, c) a list of values for the represented attributes (nominal and continuous), d) a list with the positions (or memory offset) where each attribute can be found in the previous structure and e) the class of the classifier.

ALKR Classifier Example

numAtt	3
whichAtt	0 2 4
predicates	0.5 0.7 1 1 0 0.3 0.4
offsetPred	0 2 5
class	1

Figure 2.7: Representation of a classifier using ALKR in BioHEL

Since the ALKR list usually does not represent all the attributes, the probability of an attribute to appear in a randomly generated classifier depends on the parameter *ExpAtts*. This is an user-defined parameter that determines the expected value for the number of relevant attributes in a rule. Based on this value and the number of attributes d it is possible to calculate the probability l_d of an attribute to appear in the attribute list as follows:

$$l_d = \begin{cases} 1 & d \leq \text{ExpAtts} \\ \frac{\text{ExpAtts}}{d} & d > \text{ExpAtts} \end{cases} \quad (2.4.10)$$

After determining which attributes are going to appear in the list, the predicate for each attribute is determined depending on its type. For the discrete attributes the initialisation follows the same methodology explained in Section 2.4.1.1. In the case of continuous intervals a random span s between 0.25 and 0.75 is selected. Afterwards, if using covering, the value of the instance will represent the centre c of the interval and the lower and upper bounds would be $c - s/2$ and $c + s/2$, respectively. If covering is not used then the centre c is also selected randomly. It is important to mention that when covering is used all classes have the same probability of being selected for covering despite of the class imbalance which proved to be beneficial to increase the probability of covering the whole search space [Bacardit, 2005].

In problems with both discrete and continuous attributes ALKR stores temporarily the nominal and continuous attributes of the rules separately in two different vectors during the evaluation process. Afterwards, two loops (one for each type of attribute) perform the match independently for each type. This technique improved the performance of the match of mixed attributes as shown in [Bacardit and Krasnogor, 2009a].

2.4.2.2 Fitness function

BioHEL also uses a fitness function based of the MDL principle [Rissanen, 1978] inspired in GAssist's fitness function. This fitness function promotes accurate, general and compact rules by combining three elements: accuracy, coverage and complexity of the rules [Bacardit et al., 2009a,b]. This fitness function has two terms, as it was already shown in Equation (2.4.1) but applied over a rule a instead of a rule set (since in BioHEL each individual is a rule). However, the definition of the $EL(a)$ term changes to include not only the accuracy, but also the coverage of the rules. For BioHEL, $EL(a)$ is defined as:

$$EL(a) = 2 - ACC(a) - COV(a) \quad (2.4.11)$$

$$ACC(a) = \frac{correctlyClassified(a)}{matched(a)} \quad (2.4.12)$$

$$COV(a) = \begin{cases} 0 & \text{if } RC(a) < CB(a.class)/3 \\ PenaltyCov(a) & \text{if } RC(a) < CB(a.class) \\ HighCov(a) & \text{if } RC(a) \geq CB(a.class) \end{cases} \quad (2.4.13)$$

$$PenaltyCov(a) = CR * \frac{RC(a)}{CB(a.class)} \quad (2.4.14)$$

$$HighCov(a) = CR + \frac{(1 - CR) * (RC(a) - CB(a.class))}{1 - RC(a)} \quad (2.4.15)$$

$$RC(a) = \frac{correctlyClassified(a)}{|T_{a.class}|} \quad (2.4.16)$$

$$CB(c) = CB * \frac{|T|}{|T_c|} \quad (2.4.17)$$

$ACC(a)$ corresponds to the accuracy of the rule a and $COV(a)$ corresponds to the coverage of the rule a , which is the key of BioHEL's fitness function. As shown in Equation (2.4.13), the value of $COV(a)$ depends on $RC(a)$ (recall; the ratio between the number of examples correctly classified by the rule over the total number of examples in the training set belonging to the same class as a) and $CB(a.class)$ (the percentage of examples of its class that any rule should cover to be considered a "good rule"). CB corresponds to the BioHEL parameter known as *coverage breakpoint*. The coverage breakpoint parameter is set globally in the system, and afterwards it is transformed for every class considering the class distribution. This is a very problem dependent parameter which can affect the performance of the system, and it is difficult to set up. In the original version of BioHEL, coverage ($matched(a)/|T|$) was employed instead of recall to define $RC(a)$. The change was due to recall being more robust in datasets with class imbalance. To avoid changing the name of the parameters all the original formulations have been maintained except for $RC(a)$.

Moreover, CR (coverage ratio) corresponds to the percentage of "reward" awarded to a rule with a coverage higher than the CB threshold. Figure 2.8 shows the value of the coverage term $COV(a)$, depending on the coverage of the rule. The rules that do not cover this minimal percentage of examples are penalised.

Moreover, the $TL(a)$ term in the fitness calculation also changes. In the case of BioHEL the $TL(a)$ term is calculated as follows:

$$TL(a) = \frac{\sum_{i=1}^{NA} TL_i}{NA} \quad (2.4.18)$$

where TL_i is the theory length of the attribute i and NA is the number of attributes in the

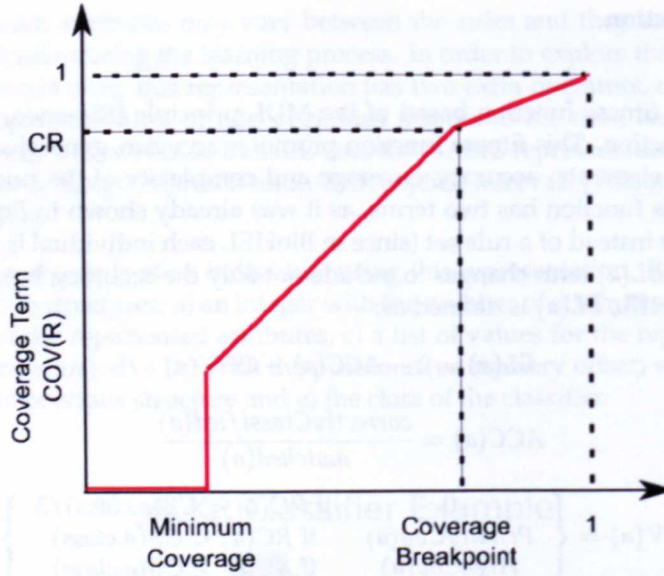


Figure 2.8: Coverage term $COV(a)$ according to rule coverage. The minimum coverage corresponds to one third of the coverage breakpoint

problem. At the same time, the definition of TL_i changes depending on whether the attribute is continuous or discrete. If the attribute is continuous the theory length is calculated as follows:

$$TL_i = \left(1 - \frac{Size_i}{MaxSize_i}\right) \quad (2.4.19)$$

where $Size_i$ is the size of the interval i and $MaxSize_i$ is the size of the corresponding domain. On the other hand, if the attribute is discrete, the theory length is calculated based on the number of zeros in the GABIL string as follows:

$$TL_i = \left(\frac{n_i}{v_i}\right) \quad (2.4.20)$$

where n_i is the number of negative values and v_i is the number of possible values of attribute i .

Regarding the weight W in the MDL fitness function, the heuristic used to adjust is exactly the same as the one used in GAssist. The only difference is that since BioHEL has individuals that are rules instead of rule sets there is no need to normalise the theory length term.

2.4.2.3 Shared mechanisms with GAssist

The mechanisms BioHEL inherit from GAssist are: a) the ILAS windowing scheme, b) the default rule mechanism, c) basic fitness function structure and c) the ensemble consensus prediction.

Both the ILAS windowing scheme and the consensus prediction using ensembles work in the same way as in GAssist, as explained in Sections 2.4.1.3 and 2.4.1.4, respectively. However, the strata are recalculated in each iteration of the IRL loop, given that the examples covered by each rule learnt by the system are removed from the training set before the next IRL iteration starts.

Moreover, as explained in Section 2.4.2.2, BioHEL's fitness function uses the same principles as GAssist fitness function, but it introduces the measure of coverage of the rules which helps promote general rules in the new learning paradigm.

Moreover, the default rule mechanism in BioHEL is a little bit different since it does not accept the automatic policy. As the rules in BioHEL are generated one at the time, it is not possible for this system to determine automatically the most favourable default class as before, since there is a lack of global supervision. Moreover, in the BioHEL system is possible to set a fixed default class.

Methodological aspects

This chapter presents the general methodological aspects that are common among the chapters throughout the thesis. This includes the statistical tests used in our experiments and the background behind the decision of using these particular tests. Furthermore, this chapter also explains in greater detail the datasets used throughout the thesis to test the performance of the analysed systems and the default parameter configurations for BioHEL, which remains fixed unless stated otherwise.

3.1 Statistical tools and performance measures

Statistical analysis is fundamental when trying to compare two or more different classifiers correctly. In this thesis, a common statistical methodology is followed throughout all chapters. The adopted methodology is based on previous studies that have aimed to determine which are the most suitable statistical tests to compare two or more classifier implementations. For instance, Demšar [2006] compared different parametrical and non-parametrical tests to establish comparisons and conclusions among two or more classifiers over multiple datasets. He started from the idea that in many occasions the results in the ML community are presented without a sound statistical basis, and this is very important when the conclusions state that one classifier is better than other. In this work, he points out at the fact that the accuracy measures do not usually comply with the restrictions to use a parametric test (independence, normal distribution and homogeneity). Therefore, this study recommends non-parametric test which are safer since they do not assume these characteristics. They recommend the Wilcoxon test [Wilcoxon, 1945] for the comparison between two classifiers and the Friedman test [Friedman, 1937] for comparisons that involve more than two classifiers. He also warns against t-test which is usually conceptually inappropriate and unsafe. As post-hoc tests this work presents the Nemenyi test [Nemenyi, 1963], the Bonferroni-Dunn test [Dunn, 1961], Holm [Holm, 1979] and Hochberg [Hochberg, 1988] while the first one can determine significant differences between each pair of classifiers at the same time, the rest are used to determine significant differences between a control method and the rest.

Afterwards, García and Herrera [2009] extended this study to deal with aspects that were not fully analysed in Demšar's paper, such as other methods to perform pairwise comparisons and the adjustment of the p-values by the post-hoc procedures. Moreover, García et al. [2009] also extended this study by comparing different accuracy measures and interpretability measures over the classifiers. The two accuracy measures presented in this work are the widely used classification rate [Witten and Frank, 2005] and the Cohen's kappa [Cohen, 1960]. The latter is very similar to the classification rate but considers the random success. On the other hand, the interpretability measures analysed are the total number of rules generated and the number of antecedents of clauses in the condition (which were adopted as the performance measures in Chapter 9). They analysed the impact of using different measures over the different tests.

First they show that the conditions necessary to apply parametric test are rarely presented. No matter the performance measure used the normality and homogeneity restrictions are seldom satisfied, and they usually depend on the problem at hand. Another important reason for using non-parametric tests is that when comparing with deterministic approaches it is difficult to obtain large amounts of samples. The conclusions are very similar to the ones in [Demšar, 2006]. Moreover, as post-hoc tests for multiple comparisons they recommend the use of Holm [Holm, 1979] and Hochberg [Hochberg, 1988] procedures which were the ones they found capable of distinguish slight differences between the algorithms.

García et al. [2010] also extended Demšar [2006] to introduce other non-parametric methods that can be used to perform comparisons between multiple classifiers. The new non-parametric techniques (different from the Friedman test) presented in this work were the Múltiple Sign-test [Steel, 1959] and the Contrast Estimation based on medians. Moreover, they also analyse other alternatives to the Friedman test, the Friedman Aligned Ranks [Hodges and Lehmann, 1962] and the Quade test [Quade, 1979]; and other post-hoc procedures such as Holland, Rom, Finner and Li. However, in this thesis there was no need to use any of this advanced techniques.

In this thesis we use the Wilcoxon test to compare two different approaches (classifier versions). To compare more than two approaches we used the Friedman test if the comparison is made across different datasets. As post-hoc tests (when comparing more than two classifiers) we use the Nemenyi test to determine significant differences without using a control method and the Holm test when a control method is ruled out. When the comparison is between more than two approaches but it is made between observations taken from the same dataset the Friedman test is not applicable. In this case we apply the Kruskal-Wallis test followed by the Wilcoxon pairwise test using the Holm correction. The following sections explain in greater detail how these statistical tests work.

3.1.1 Wilcoxon signed-ranked test

The Wilcoxon signed-ranked test [Wilcoxon, 1945] is a non-parametric test used in this thesis to compare two classifiers. It ranks the difference in performance of the classifiers in each dataset, separating the positive and negative differences. This test is the non-parametric alternative of the commonly used t-test.

For this test the data should be paired, and each pair should correspond to an independent and randomly obtained observation. The null hypothesis of this test states that the mean difference between the two samples is zero. The statistic W is calculated as:

$$W = \sum_{d_i \neq 0} \text{rank}(|d_i|) \cdot \text{sgn}(d_i) \quad (3.1.1)$$

$$d_i = x_{1,i} - x_{2,i} \quad (3.1.2)$$

where $\text{rank}()$ is a function that ranks all the absolute values of the differences between the two samples (and if there are ties awards the average ranking to both positions) and $\text{sign}()$ is a function that is equal to 1 if the difference was positive and equal to -1 if the difference was negative.

3.1.2 Friedman test

The Friedman test [Friedman, 1937] is the non-parametric alternative to the ANOVA test for normalised data. It is used to compare more than two classifiers at the time. Having a number N of observations over k classifiers, this test ranks the classifiers per each observation block. Then it compares the average ranks of the classifiers which is calculated as follows:

$$R_j = \frac{1}{N} \sum_i r_{i,j} \quad (3.1.3)$$

where $r_{i,j}$ is the rank of the classifier j in the observation i . The null hypothesis of this test states that the algorithms have equal ranks and the statistic (for $N > 10$ and $k > 5$) is calculated as follows:

$$\chi_F^2 = \frac{12N}{k(k+1)} \left(\sum_j R_j^2 - \frac{k(k+1)^2}{4} \right) \quad (3.1.4)$$

After the null hypothesis is rejected a post-hoc test can be applied to determine significant differences among the algorithms. When there is no need of comparing all algorithms against a control method the Nemenyi test is used in this thesis and on the opposite scenario the Holm post-hoc procedure is applied.

3.1.2.1 Nemenyi test

The Nemenyi test [Nemenyi, 1963] is a post-hoc test used to determine significant differences among each pair of algorithms without considering a control method. This test is similar to the Turkey test for ANOVA. According to this test the performance of two classifiers is significantly different if their ranks R_j differ at least by the critical distance which is calculated as follows:

$$CD = q_\alpha \sqrt{\frac{k(k+1)}{6N}} \quad (3.1.5)$$

where q_α are critical values which depend of the number of classifiers and the significance level α and they can be found in [Demšar, 2006].

3.1.2.2 Holm post-hoc procedure

The Holm post-hoc test [Holm, 1979] it is used in this thesis to determine significant differences between a control method and the rest of the methods. This test orders the null hypotheses of the comparisons by their p-value p_i from smallest to largest. Then if $p_i < \alpha/k-i$ the comparison i is statistically significant with a level of confidence α . The moment the inequality does not hold all the subsequent hypotheses with larger p-values are automatically discarded. The Hochberg procedure [Hochberg, 1988] is very similar but works in the opposite direction comparing the hypotheses from largest p-value to lowest.

3.1.3 Kruskal-Wallis test

The Kruskal-Wallis test [Kruskal and Wallis, 1952] is it used in this thesis to perform comparisons between different classifiers when the observations come from a single dataset. This a non-parametric method to test if more than two samples originate from the same distribution. It can be considered an extension of the Wilcoxon test already explained.

This method ranks all the observations from all classifiers together. Then assigns any tied values the average of the ranks they would have received had they not been tied. The statistic for this test is:

$$K = (N - 1) \frac{\sum_{i=1}^g n_i (\bar{r}_i - \bar{r})^2}{\sum_{i=1}^g \sum_{j=1}^{n_i} (r_{ij} - \bar{r})^2} \quad (3.1.6)$$

where g is the number of classifiers or groups, N is the total number of observations, n_i is the number of observations in group i , r_{ij} is the rank of the observation j from group i , \bar{r}_i is the average rank of group i and \bar{r} is the average of all the ranks, which is equal to $1/2(N + 1)$.

The null hypothesis of equal population medians would then be rejected if $K \geq \chi_{\alpha;g-1}^2$.¹ If significant differences are found a Wilcoxon pairwise test with Holm correction for multiple comparisons can be applied to determine these differences.

3.2 Datasets

Different types of datasets are used throughout the thesis to test the system with different purposes. The aim of this section is to present the characteristics of these datasets and its classification.

There are two main types of datasets used in this work: real-world datasets and synthetic datasets. The difference between the two is that the latter were created with specific characteristics that allows us to gather particular insights of the learning process. Moreover, since the characteristics of the real datasets are very different, we separate them in two big groups (big and small datasets) which serve different purposes. The following sections will explain these two types of datasets.

3.2.1 Real-world datasets

Table 3.1 presents a list of the characteristics of the datasets used. The real datasets are only used in Chapters 4, 7, 8 and 9, since the other two chapters present empirical analyses and theoretical formulations that required the use of synthetic benchmarks. Since some chapters only use a subset of them, this table shows a \star if the dataset was used in that particular chapter. Furthermore, these datasets can be classified in two big groups: small and large datasets. In this thesis we consider small datasets those which have less than 40000 instances. All datasets are used to test BioHEL's prediction capacity, but the big ones are also used to test the scalability capacity of the system. It is worth mentioning that the concept of "large datasets" has been evolving really fast in recent years. Even though there exist really big datasets nowadays, the ML methods commonly used over these problems are very simple. The reason for using these sizes of datasets in this thesis is because the aim is to test the scalability boundaries of more sophisticated ML approaches. Moreover, some of these real benchmarks have been already used in the past to test other characteristics of BioHEL, which makes it easier to compare with previous states of the algorithm.

All the small datasets and the Adult (Adu), connect-4 (c-4) and KDDCup (kdd) problems were taken from the UCI repository of ML datasets [Blake et al., 1998]. The CN, CN0, CN-bin, SS and SA are Protein Structure Prediction (PSP) datasets compiled by the ICOS group at the University of Nottingham as part of its research in bioinformatics.² These datasets are interesting and challenging because of the large number of instances and attributes and the high levels of uncertainty they present. These datasets have been used already in [Bacardit et al., 2009a, 2006; Bacardit and Krasnogor, 2008b, 2009a; Stahl et al., 2010]. The ParMX (par) dataset is a hybrid

¹Exact values should be used when there less than 5 observations per group

²The complete benchmark suite can be accessed at http://icos.cs.nott.ac.uk/datasets/psp_benchmark.html

Table 3.1: Characteristics and classification of the datasets used in this thesis. #Ins = Training set size, #Att = Number of attributes, #Dis = Number of discrete attributes, #Con = Number of continuous attributes, #Cl = Number of classes. * means the dataset was used in that particular chapter

	Name	#Ins	#Att	#Dis	#Con	#Cl	Chapters			
							4	7	8	9
Small Datasets	bal	559	4	0	4	3	*			*
	bpa	311	6	0	6	2	*			*
	bre	257	9	9	0	2	*			*
	cmc	1327	9	7	2	3	*			*
	col	332	22	15	7	2	*			*
	cr-a	622	15	9	6	2	*			*
	glc	193	9	0	9	6	*			*
	h-cl	274	13	7	6	2	*			*
	hep	139	19	13	6	2	*			*
	h-h	263	13	7	6	2	*			*
	h-s	243	13	0	13	2	*			*
	ion	316	34	0	34	2	*			*
	irs	135	4	0	4	3	*			*
	lab	52	16	8	8	2	*			*
	lym	133	18	15	3	4	*			*
	pen	9892	16	0	16	10	*		*	*
	pim	691	8	0	8	2	*			*
	prt	300	17	17	0	21	*			*
	sat	5792	36	0	36	6	*		*	*
	son	187	60	0	60	2	*			*
	thy	194	5	0	5	3	*			*
	vot	391	16	16	0	2	*			*
	wav	4539	40	0	40	3	*		*	*
	wbcd	630	9	0	9	2	*			*
	wdbc	510	30	0	30	2	*			*
	wine	159	13	0	13	3	*			*
	wpbc	176	33	0	33	2	*			*
	zoo	91	16	16	0	7	*			*
Large	SS	75583	300	0	300	3	*		*	*
	Adu	43960	14	8	6	2	*		*	*
	c-4	60803	42	42	0	3	*		*	*
	far	90868	29	24	5	8			*	
	PMX	235929	18	18	0	2	*		*	*
	kdd	444619	41	15	26	23	*		*	*
	SA	493788	270	26	244	2	*		*	
	CN0	234638	20	0	20	2			*	
	CN-bin	234638	0	10	10	2		*	*	*
	CN	234638	180	0	180	2	*		*	*

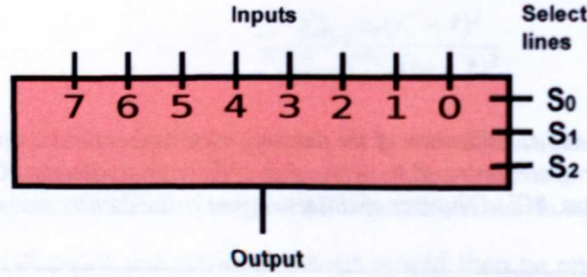


Figure 3.1: Multiplexer with 8 inputs (string bits) and 3 select lines (address bits).

parity-multiplexer dataset already proposed by [Butz, 2006]. Finally, The FARS - Fatality Analysis Reporting System dataset (far) was taken from the U.S National Highway Traffic Safety Administration.

3.2.2 Synthetic datasets

In Chapters 5, 6, 7 and 8 synthetic datasets were used because there was a need to test the system under a controlled environment, under which either the correct answer (optimal set of rules) is known and/or the degree of difficulty may be varied on a regular step. The synthetic datasets used are the k-Disjunctive Normal Form (k-DNF) family of problems and the multiplexer problem.

The k-DNF problems are boolean formulas generated randomly which have r disjunctive terms, with k variables expressed each out of the d possible variables. Some of these variables can have the \neg function applied to them. For example a random k-DNF function with $r=3$, $k=2$ and $d=5$ would be:

$$(x_1 \wedge x_2) \vee (\neg x_2 \wedge x_4) \vee (\neg x_1 \wedge x_5)$$

The importance of this family of datasets is that it is possible to generate randomly different datasets with the same characteristics (k , r and d), and test the scalability of the system, not over a particular problem, but over a particular set of problem characteristics. Also these formulas are very useful to characterise boolean problems as it will be shown latter in this thesis. The generation of k-DNF datasets and the theory around them will be explained in greater detail in Chapter 5.

In Chapter 6 also multiplexer and ternary multiplexer problems were used. The concept of multiplexer is taken from the electronic field and means a device which has 2^n analog inputs, n select lines and 1 output. The select lines determine which of the inputs will be passed as the output. The combination of the select lines is usually called the *address* while the rest of the bits are usually called the *string*. Figure 3.1 shows an example of multiplexer with 8 string bits and 3 address bits.

The ternary multiplexer is a variation of the original multiplexer problem used in this thesis to test the BioHEL system in χ -ary domains. The only difference is that instead of each one of the input bits in the string and the address being boolean, this problem accepts three values $\{0,1,2\}$. This means that for each n bits in the address we will have 3^n inputs.

Table 3.2: Fixed parameters for the BioHEL system

Parameter	Value
Population size	500
GA Iterations	50
Repetitions of rule learning (GA repetitions)	2
Default class	majority
Number of windows in ILAS	2
<i>Operators</i>	
Crossover operator	1 point
Prob. crossover	0.6
Prob. individual mutation	0.6
Selection algorithm	tournament
Tournament size	4
<i>MDL Fitness function</i>	
Coverage breakpoint <i>CB</i>	<problem dependant>
Coverage ratio <i>CR</i>	0.90
Initial MDL TL ratio	0.025
Iterations to change MDL weight	10
MDL weight relax factor	0.90
<i>ALKR Representation</i>	
Prob. one <i>p</i>	0.75
Expressed Attributes (<i>ExpAtts</i>)	15
Prob. generalize the ALKR list	0.10
Prob. specialize the ALKR list	0.10
Covering	activated
Balanced class distribution	activated

3.3 Basic parameter configuration for BioHEL

Along the thesis, tests are ran over BioHEL with different purposes in which several parameters of the system are modified. However, there is a main configuration of the system that remains fixed, which is tied to the aspects of the system that are not analysed in this thesis. Table 3.2 shows the configuration of the system for the experiments in this thesis, that remains the same if not stated otherwise.

For GAssist, the configuration used is presented in Chapter 4 which is the chapter in which this system is analysed.

Pittsburgh vs. IRL: Identifying future challenges

This chapter reports an exhaustive analysis performed over BioHEL and its predecessor GAssist. These two systems share many mechanisms and operators, but at the same time, they apply two different learning paradigms (the IRL approach and the Pittsburgh approach, respectively). The aim of this analysis is to: a) propose standard configurations for handling small and large datasets, b) compare the two systems in terms of learning capabilities, complexity of the obtained solutions and learning time, c) determine the areas of the problem space where each one of these systems performs better, d) compare them with other well-known ML algorithms, and e) determine limitations and possible areas of improvement in BioHEL. The results show that it is possible to find standard configurations for both systems. With these configurations the systems perform up to the standards of other state-of-the-art ML algorithms such as SVM. However, when the systems are parameterised according to the problem characteristics better results can be obtained. Moreover, we identify the problem domains where each one of these systems have advantages and disadvantages and propose ways to improve BioHEL based on this analysis.

4.1 Introduction

Many different types of EL systems exist. Some of these systems, even though they use different learning paradigms (i.e Michigan, Pittsburgh, IRL), might share some mechanisms such as representations, fitness functions, efficiency enhancement techniques, among others. Therefore, it is important to compare the systems among themselves to determine if the shared mechanisms can be used in the same way despite the change of paradigm. Moreover, this type of comparisons helps us understand the advantages and disadvantages of the systems (as well as their domains of competence) in order to improve and make better use of them. Finally, this type of analysis is particularly important in the context of this thesis, because it is the start point to determine weaknesses of the BioHEL system and the possible areas of improvement.

This chapter presents a thorough comparison between the BioHEL system, which is the focus of study of the thesis, and its predecessor GAssist (see Sections 2.4.2 and 2.4.1, respectively). As it was explained previously, these two systems share several mechanisms, such as smart initialisation operators, knowledge representations and efficiency enhancement approaches. The main difference is that BioHEL uses IRL while GAssist is a Pittsburgh-style LCS. Therefore, many questions arise. Can each of the shared components be used in the same way in systems with different paradigms? Do these components affect each system in a very particular way? Can these systems be applied to the same type of problems? Do these systems scale in a similar way? How competitive are these systems compared to other state-of-the-art ML algorithms? Where are the areas of improvement for BioHEL at this point? The aim of this chapter is to answer these questions and get more insights into the advantages and disadvantages of the

use of these two systems in different domains. We focus our efforts on determining a parameter configuration for these systems, that works acceptably well on a broad range of problems, and assessing the gap between these global settings and the problem-dependent optimal settings.

We analyse the impact of the policy selection in the default class mechanism, which is a mechanism shared among the two, tangentially to the two most important parameters in both systems. In the case of BioHEL, this is the coverage breakpoint parameter; and in the case of GAssist, the number of GA iterations. Afterwards, we analyse in deep the ILAS windowing scheme (see Section 2.4.1.3), which is the mechanism that allows both systems to scale successfully in problems with large training set sizes. Finally, we compare the two algorithms with other well-known ML mechanisms.

The experiments showed that it is possible to find a standard configuration for both systems that avoids the cost of hand tuning parameters and performs well in most problems. However, the BioHEL system is more sensitive to the usage of a standard configuration than GAssist, and specially to the choice of coverage breakpoint. Moreover, it is shown how the usage of windows increases the scalability and generalisation capabilities of the systems, but also affects the prediction capacity of both algorithms in terms of accuracy. Our results show how, by means of using different sets of parameters, the user can find the right balance between generalisation and accuracy to obtain good and compact solutions in less time. Also, the comparison with other ML algorithms shows that our systems perform as good as SVM and better than other well-known algorithms.

This chapter is organised as follows. Section 4.2 presents the existent comparisons between GBML systems in the literature. Section 4.3 discusses the reasons why this chapter performs a parameter sensitivity analysis instead of applying other parameter setting techniques. Section 4.4 presents the experimental design for the comparison. Section 4.5 presents the results separated in four stages: a) parameter sensitivity analysis, b) comparison of BioHEL vs. GAssist, c) analysis of the ILAS windowing scheme and d) comparison with other ML techniques; and finally, Section 4.6 presents the final remarks and the discussion about possible improvements in the BioHEL system.

4.2 Previous comparisons between GBML systems

Several comparisons have been made in the past between GBML systems. For instance, comparisons were made between XCS and two Pittsburgh-style LCS, GALE and GAssist. The comparison between GALE and XCS, performed by Bernadó-Mansilla et al. [2006], is centred around the performance of the algorithms, while the comparison between XCS and GAssist, made by Bacardit and Butz [2007], pointed out the advantages and disadvantages of the systems when solving particular problems. Particularly, in the latter work, the results were presented not only in terms of accuracy, but also in terms of number of rules generated. The results showed that GAssist had some problems handling too many classes while XCS had problems because it tends to overfit the data.

A similar analysis was performed by Orriols-Puig et al. [2008a,b] comparing UCS and GAssist. In terms of performance (test accuracy), the UCS system obtained the best results, while in terms of interpretability of the solution, GAssist generated the most manageable models.

Comparisons between EL and other ML algorithms have also been carried out in [Bernadó-Mansilla et al., 2006; Fernández et al., 2010; Orriols-Puig et al., 2008a,b]. In these experiments, the genetic-based approaches proved to be good candidates to solve classification problems when compared with other well-known data mining techniques, such as NaïveBayes [John and Langley, 1995], C4.5 [Quinlan, 1993], PART [Frank and Witten, 1998], IBk [Aha et al., 1991], SMO [Platt, 1999], Fuzzy AdaBoost [Alcalá-Fdez et al., 2009] and Fuzzy LogitBoost [Alcalá-

Fdez et al., 2009]. Particularly in [Orriols-Puig et al., 2008a] GAssist is included in the set of algorithms to compare, but without using ensembles which produced that this algorithm ranked differently compared to the results in this chapter.

Moreover, comparisons have been performed between EL systems from a statistical point of view in [García et al., 2009]. This work was focused on establishing a statistical methodology for the comparisons between classifiers. They analyse the performance measures used nowadays and propose other performance measures for two-class and multi-class problems.

4.3 Parameter sensitivity analysis vs. self-adaptation

An often used approach to determine parameter settings in EL (particularly in Michigan LCS) is self-adaptation [Bull and Hurst, 2000; Butz et al., 2008c; Hurst and Bull, 2001a, 2002]. However, in this particular case it is not possible to apply such approach for many reasons. First, it is not possible to apply self-adaptation over the coverage breakpoint because this parameter affects the fitness directly and it will converge to a value that maximises the fitness without considering the characteristics of the problem. The less restrictive the coverage breakpoint is the higher the chances that an overgeneral rule has a higher fitness than an accurate one. Self-adapting the coverage breakpoint then will tend to promote extremely general rules which maximise the fitness, giving little consideration to their actual accuracy.

Moreover, the number of iterations in GAssist is a global parameter of the system instead of a particular parameter of the classifiers, therefore it cannot be self-adapted. The default class is also a global parameter of the system, and even though it is indeed adapted in GAssist (using the *automatic* policy), in BioHEL this is not possible because this system does not have a global overview of the generated rule sets. Finally, the number of ILAS windows could be adapted online during the GA run. However, it is not completely clear if this would be beneficial. Adapting the number of windows implies regenerating the windows each time, which would increase greatly the computational cost, and important information, such as the elite individuals of each window, would need to be discarded. This could affect the performance of the learning process. Considering this, in this thesis we decided to instead perform a thorough parameter sensitivity analysis above the variables mentioned above.

4.4 Experimental design

The experiments in this chapter are divided in four phases: a) parameter sensitivity analysis, b) comparison of GAssist with BioHEL, c) analysis of the windowing scheme and d) comparison against other ML techniques. This analysis is made using 35 real world datasets already presented in Section 3.2, which can be classified in two groups: small datasets, with less than 40000 instances, and large datasets, with up to 500000 instances. This makes it essential to make a distinction parameter-wise between the two groups.

The parameter sensitivity analysis will be focused on trying to find a standard configuration for both systems that works well with the majority of the problems, as well as analysing some of the shared mechanisms. In the case of the BioHEL system, we will analyse the coverage breakpoint parameter, and in the case of GAssist, the number of iterations of the GA. In this section we will also analyse different configurations for the default class mechanism incorporated in both systems. We analyse this mechanism in this section and not independently because this parameter can change the characteristics of the problem to solve, thus it is strongly related to the other configuration parameters. Moreover, we analyse the coverage breakpoint parameter and the number of GA iterations because these are the most influential parameters for BioHEL and GAssist, respectively. Afterwards, the standard configurations will be compared with the best possible configuration found for each problem. This will show how far are the results

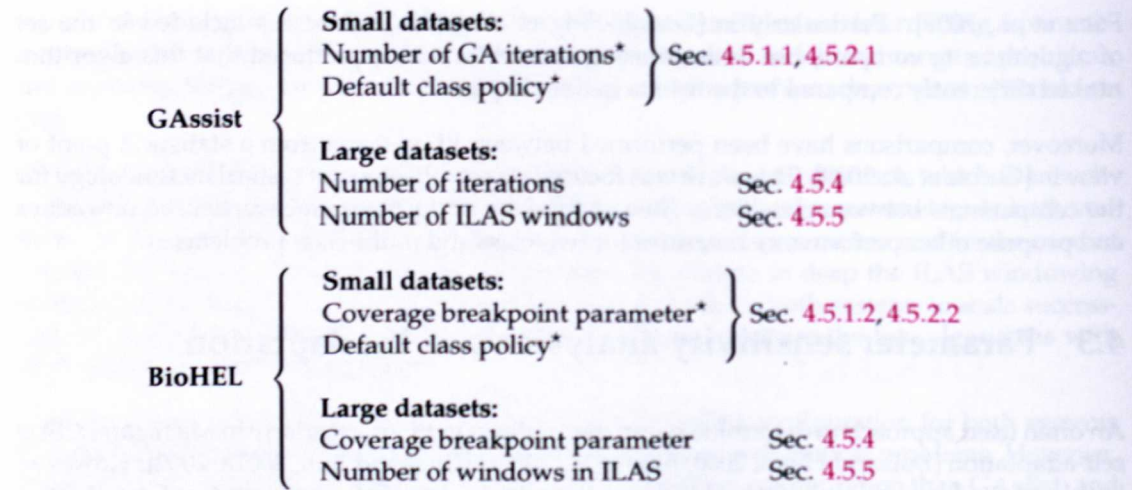


Figure 4.1: Summary of parameters analysed per system and their corresponding results sections.

*These parameters are linked to each other in the systems, hence their parameter-sensitivity analysis is combined

using a standard configuration from the best results found if we hand-tune the system. Moreover, we show an exhaustive comparison between the two paradigms in terms of performance, scalability and complexity of the solutions. Afterwards, a similar but less extensive analysis will be performed independently over large problems, since the configurations used for these two types of problems are different.

Afterwards we perform the analysis of the ILAS windowing scheme, which will be focused on analysing the behaviours of the system when the number of windows increase. In this stage we analyse the accuracy and number of rules against the execution time. Moreover, this analysis is performed only over the large datasets, since this is the kind of datasets over which the windowing would have a big impact in terms of execution time. To summarise, Figure 4.1 shows the parameters to analyse in both systems and its corresponding results section.

Furthermore, in the last phase, we compare our two systems with other well-known ML algorithms. The comparison is made with tools taken from the WEKA package [Hall et al., 2009] such as C4.5, PART, IBk, Naïve Bayes and the SMO implementation of SVM with both polynomial and Gaussian kernels. For the SVM and the IBk algorithms we performed a preliminary statistical analysis to determine the best configuration for the problems we wanted to solve. For polynomial SVM we tested kernels of order 1, 3, 5 and 10, resulting significantly better than the rest using order 1. Moreover, for SVM with Gaussian kernels we tested $\gamma = \{1, 0.5, 0.1\}$, where $\gamma = 0.5$ resulted significantly better. The same process was followed with the IBk where we tested $k = \{3, 5, 7\}$. The best results were obtained with $k = 5$.

The datasets used in these experiments are diverse. Therefore, they were separated in two groups: small and big datasets. At the same time, the later group is separated into two categories: medium and large problems. Medium problems are the ones over which we could perform the complete set of experiments. The large problems correspond to two very large problems over which we could not test all the configurations because either the execution times were impractical (about 10 days per run) or the memory requirements were too demanding. Table 4.1 contains a list of the datasets that fall on each category. For a complete description of the datasets please see Section 3.2.

The experiments reported in this chapter were performed in the High Performance Computing facility at the University of Nottingham provided with 2 x Intel Xeon (Harpertown) 3.0 GHz quad core with 2GB per core. Moreover, the experiments with the WEKA package that were memory bounded were run in a dedicated machine with 4 x Intel(R) Core(TM) i7 CPU 3.07GHz

Table 4.1: *Datasets used to compare GAssist and BioHEL and its classification*

Small	bal, bpa, bre, cmc , col, cr-a, gls, h-c1, hep, h-h, h-s, ion, irs, lab, lym, pen, pim, prt, sat, son, thy, vot, wav, wbcd, wdbc, wine, wpbc, zoo
Medium	Adu, c-4, PMX, kdd
Large	SA, CN

with hyper threading and 12GB of RAM.

4.4.1 Evaluated scenarios

Both BioHEL and GAssist depend on parameters set beforehand by the user. In the case of BioHEL the most influential parameter is the coverage breakpoint. For small problems, we tested the coverage breakpoints $CB = \{0.5, 0.25, 0.1, 0.05, 0.01\}$. On the other hand, for larger problems the coverage breakpoints tested are smaller $CB = \{0.1, 0.05, 0.01, 0.005, 0.001\}$. This distinction was made due to the fact that in small problems each rule tends to cover a large percentage of the search space while for large problems the solution tends to be more complex and fine-grained.

The rule sets generated with GAssist depend on the number of GA iterations the system runs. The number of iterations should be large enough to converge to a good solution but small enough to avoid overfitting the data. Small problems converge faster, so for this type of problems we tested 100, 250, 500, 1000 and 1500 iterations. For larger problems, we tested slightly larger numbers than before, such as 1000, 2500, 5000 and 10000 iterations.

Regarding the default class mechanism, for the BioHEL system we tested the majority, minority and disabled policies. In the GAssist system, we tested the same policies as in BioHEL plus the automatic policy, which cannot be used in BioHEL as it requires the whole rule set, hence global supervision. For the big problems, only the majority default class policy was tested to reduce the global execution time of the experiments. The majority class was chosen for these problems because, as it was shown in the past [Bacardit, 2004], not allowing to generate rules for the majority of the examples, reduces the number of examples that need to be fitted by rules which could translate in a smaller execution times.

For the parameter sensitivity analysis over the coverage breakpoint and the default class mechanism, the number of windows remained fixed. For small problems we used only 2 windows while for large problems we used 10 windows. Afterwards, the impact of the number of windows is analysed using the best configurations found on the first stage. The rest of the parameters for BioHEL and GAssist remain fixed to the values shown in Table 4.2.

For the analysis of the ILAS windowing scheme only big datasets were used. We tested scenarios using a number of windows $w = \{10, 20, 50, 100, 200\}$ and we observed the test accuracy, number of rules and execution time. During these experiments the rest of the parameters of the system remained fixed as before. Finally, for the comparison with other ML techniques only the best standard configuration found for GAssist and BioHEL was used.

4.4.2 Performance metrics and statistical analysis

The results in the parameter sensitivity stage and the ILAS windowing scheme analysis are presented in terms of ensemble test and training accuracy, execution time and number of rules

Table 4.2: Parameter configuration for GAssist and BioHEL

		BioHEL	GAssist
Shared parameters	Population size	500	500
	Crossover	1 point	1 point
	Probability of crossover	0.6	0.6
	Probability of mutation	0.6	0.6
	Probability of one p	0.75	0.9
	Selection algorithm	tournament	tournament
	MDL initial ratio	0.25	0.075
	MDL iteration	10	25
	MDL weight relax factor	0.90	0.90
	Tournament size	4	3
	Class wise initialisation	yes	yes
	Smart initialisation	yes	yes
BioHEL	GA iterations	50	-
	Repetitions of rule learning	2	-
	Coverage ratio	0.90	-
	Expressed attributes	15	-
	Probability of generalise	0.1	-
	Probability of specialise	0.1	-
GAssist	Probability of merge	-	0.05
	Probability of split	-	0.05
	Probability of reinitialise	-	0.03
	Probability of reinitialise at end	-	0.00
	Pruning iteration	-	5
	Pruning minimum classifiers	-	12
	Penalise individuals less than	-	4 classifiers

generated.¹ The test and training accuracies provide insights into the learning capabilities or over-fitting problems of the systems. Moreover, the execution time will provide a comparison in terms of the scalability of the systems. Finally, the number of rules will compare the systems in terms of how simple, compact and general were the solutions found. When comparing with other ML methods only the test accuracy and the execution time will be considered.

For all our experiments we used a ten-fold cross validation. We performed 25 repetitions over small datasets and 5 over large ones. Moreover, ensemble techniques were used to compute the final accuracy per fold as explained in Section 2.4.1.4.

For the statistical analysis we follow the methodology described in Section 3.1. To determine significant differences between more than two scenarios (different configurations) across several datasets we used the Friedman test. If there are significant differences, we use the Holm post-hoc test to find these particular differences between the control method and the rest. For the analysis of the best problem-dependent configuration the Friedman test cannot be used as the configurations are compared over non-independent observations over one single problem. Therefore, in this case the Kruskal-Wallis test was used to determine significant differences among the configurations and, where significant differences were encountered, the pairwise Wilcoxon test was used to compare the control against the rest of the configurations. Moreover, to determine differences between the best configuration found per problem and the standard configuration, we used the one-tailed Wilcoxon signed-rank test. This test was applied both per problem and over all datasets at the same time. For the comparison between our algorithms and the WEKA package we used the Friedman test, followed by the analysis of

¹Memory requirements are not reported in this chapter since the memory is mostly dominated by the size of the training sets instead of the solutions generated

the Nemenyi critical distance to determine significant differences between all the algorithms among each other.

4.5 Results

In this section we present the results of comparing BioHEL with GAssist. The first two sections perform a parameter sensitivity analysis over small problems, to determine global and problem-specific configurations. The next section presents a thorough comparison between GAssist and BioHEL in terms of performance, scalability and complexity of the solutions. Afterwards, a similar, but less extensive analysis is performed over large scale datasets to determine a global configuration in these cases. Later on, an analysis over the ILAS windowing scheme is performed to determine the impact of this shared mechanism on each one of the systems. Finally, we show how these two systems, using a standard configuration, compare with other ML algorithms taken from the WEKA package.

4.5.1 Finding a global standard configuration

In this stage we analyse how the coverage breakpoint (for BioHEL) and the number of GA iterations (for GAssist) affect the accuracy obtained with different default class policies in both systems. The procedure used in the following sections to analyse the different configurations in BioHEL and GAssist is the following:

1. The default class and a system dependant parameter (number of GA iterations in the case of GAssist and coverage breakpoint in the case of BioHEL) was varied to construct different configuration scenarios.
 - (a) For each scenario the ensemble test accuracy was computed for each small dataset on Table 4.1 using 10-fold validation and 25 replications (please see Section 4.4.2)
 - (b) Using the accuracy as a performance metric, the different runs corresponding to different configuration scenarios are ranked in three different ways (ranking schemes):
 - i. within a particular default class
 - ii. within a particular system parameter
 - iii. overall ranking (comparing all configurations against each other)
 - (c) The average rank was computed per configuration scenario.
2. The Friedman test was applied to find significant differences between configurations using the average rankings for each ranking scheme.
 - (a) The average rankings per configuration are shown.
 - (b) Best ranked configuration is shown in bold and used as control.
 - (c) When significant differences are found a post-hoc test is applied to compare the control with the rest of the configurations.

This section is focused on finding relationships between these parameters and determining a standard configuration for both systems. In this section we will only show the results of the statistical tests. The complete set of results can be found in Appendix A.

Table 4.3: Average rankings and p-values of the Friedman test to determine the best number of iterations per default class policy in GAssist

Default Class	Number of iterations for GAssist					p-value
	100	250	500	1000	1500	
Majority	3.12	2.87	2.92	3.10	2.96	0.9644
Minority	2.87	3.01	2.82	3.25	3.03	0.8591
Disabled	3.26	3.32	3.05	2.57	2.78	0.2991
Automatic	3.42	3.28	3.08	2.44	2.75	0.1043

Table 4.4: Average rankings and p-values of the Friedman test to determine the best default class per number of iterations in GAssist

Iterations	Default class policies for GAssist				p-value
	Majority	Minority	Disabled	Automatic	
100	2.48	2.50	2.57	2.44	0.9851
250	2.66	2.46	2.41	2.46	0.8893
500	2.44	2.35	2.55	2.64	0.8416
1000	2.58	2.67	2.41	2.32	0.7007
1500	2.66	2.44	2.64	2.25	0.5779

4.5.1.1 GAssist

Table 4.3 shows the comparison (average rankings) of the number of iterations using different default class policies in GAssist. The Friedman test was applied to each one of the default class policies independently. In the last column of this table (p-value) we can observe that for GAssist there are no significant differences between using different numbers of GA iterations. Nevertheless, we can observe that, on average, the disabled and automatic policies need more iterations to obtain slightly better results, while the majority and minority policies seem to need less. This is due to the fact that learning without a default class policy or trying to adjust it during the learning process means exploring a larger search space, and hence the need for more GA iterations.

Table 4.4 presents the inverse analysis: for each one of the number of GA iterations we apply the Friedman statistical test, to determine which one is the most convenient default class policy. This table shows that for GAssist there are not significant differences between using different default class policies.

To analyse further the dependency between the default class policies and the number of iterations, we performed the Friedman test to compare all the different configurations at the same time. In the case of GAssist, this consist in comparing 20 different scenarios as shown in Table

Table 4.5: Average rankings of the Friedman test over all the configurations in GAssist (p-value 0.5508).

Def. Class	Number of iterations				
	100	250	500	1000	1500
Majority	12.00	11.46	10.44	11.16	10.58
Minority	10.48	10.82	9.44	10.75	9.80
Disabled	11.98	11.53	11.00	9.78	10.12
Automatic	11.30	10.75	10.10	7.87	8.57

Table 4.6: Average rankings and p -values of the Friedman test to determine the best coverage breakpoint per default class policy in BioHEL. The last column of the table shows the significant differences determined by the post-hoc test with $\alpha = 0.05$

Def. Class	Coverage breakpoints					p-value	Relations ($\alpha = 0.05$)
	0.5	0.25	0.1	0.05	0.01		
Majority	3.50	3.16	2.58	2.60	3.14	0.1287	
Minority	3.75	2.83	2.21	2.66	3.53	0.0009	0.1 > 0.5, 0.01
Disabled	3.58	3.05	2.42	2.66	3.26	0.0447	0.1 > 0.5

Table 4.7: Average rankings and p -values of the Friedman test to determine the best default class for each coverage breakpoint in BioHEL. The last column of the table shows the significant differences determined by the post-hoc test with $\alpha = 0.05$

Cov. break	Default class policies (BioHEL)			p-value	Relations ($\alpha = 0.05$)
	Majority	Minority	Disabled		
0.5	2.18	1.86	1.96	0.4353	
0.25	2.23	2.09	1.68	0.0929	
0.1	2.16	1.89	1.95	0.5610	
0.05	2.30	2.12	1.57	0.0157	<i>Dis > Major, Minor</i>
0.01	2.21	2.09	1.70	0.1223	

4.5. In this case no significant differences were found between the different scenarios (p -value 0.5508). This is consistent with the findings in the previous tables where no configurations stand out significantly. At this point we could argue that the best parameters in GAssist are either strongly problem dependent, thus it is more difficult to find a standard configuration that works well with all the datasets, or there are no significant differences in general between the presented scenarios. The next sections will be focused on proving or discarding each one of these hypotheses while analysing the configurations per problem.

Nevertheless, the scenario with the best ranking was the automatic policy with 1000 iterations. Even though this is not significantly better than the others, it is going to be used for further analysis in the following sections to quantify the drawbacks of using a static configuration for all problems.

4.5.1.2 BioHEL

Table 4.6 shows which is the more favourable coverage breakpoint value for each default class policy. The Friedman test was applied to each one of the default class policies independently. If there are significant differences, the rightmost column of the table shows the relationships found by the post-hoc Holm test. We can see in this table that the preferred coverage breakpoint by all the default class policies is 0.1. On the contrary to the observations over GAssist, BioHEL does not seem to need different coverage breakpoints for the different default class policies. Moreover, for the minority and disabled policies we observe that using a coverage breakpoint of 0.1 is significantly better than using a very general or a very specific parameter value.

Table 4.7 presents the inverse analysis to determine which default class performs better with each coverage breakpoint. In one case we found that the disabled policy performs significantly better than the majority and minority policies. Also the disabled class is the one that obtains the best ranking in 3 of the 5 coverage breakpoints values tested.

To determine if there is a relationship between the coverage breakpoint parameter and the

Table 4.8: Average rankings of the Friedman test over all the configurations in BioHEL (p -value 5.607×10^{-5}). The bottom part of the table shows the significant differences determined by the post-hoc test with $\alpha = 0.05$

Default Class	Coverage breakpoints				
	0.5	0.25	0.1	0.05	0.01
Majority	10.05	9.14	7.35	8.07	9.30
Minority	9.96	7.67	5.75	7.30	9.41
Disabled	9.32	7.23	5.53	6.08	7.78

Relations ($\alpha = 0.05$)
 $Dis_{0.1} > Major_{0.5}, Minor_{0.5}, Dis_{0.5}, Minor_{0.01}, Major_{0.25}, Major_{0.01}$

default class in BioHEL, we performed the Friedman test over the 15 possible configurations. Table 4.8 shows the rankings of each configuration for BioHEL. According to this table, for small problems the best scenario was the disabled policy with a coverage breakpoint of 0.1. This was expected for the BioHEL system because, since it uses IRL. Therefore, discarding one class in small problems could force the system to learn a more difficult problem. Instead of learning the most simple rules, the system is pushed to learn rules from a subset of classes. Moreover, this is also consistent with Table 4.6, which shows that when using the disabled policy, the preferred coverage breakpoint was 0.1. This configuration is also statistically better than other configurations like $Major_{0.5}$, $Minor_{0.5}$, $Dis_{0.5}$, $Minor_{0.01}$, $Major_{0.25}$, $Major_{0.01}$ that use either a very large or very small coverage breakpoints.

The fact that some configurations performed significantly better than others suggests that the BioHEL system is more sensitive to the selection of parameters than GAssist. This could be due to the fact that the coverage breakpoint is a parameter of the system that is closely related to a problem characteristic: the size of the rules.

In this section we have shown that for GAssist there are not significant differences between the different configurations. However, the usage of the Automatic and Disabled default class policies demands larger number of iterations than the rest of the policies. In the case of BioHEL, the usage of a coverage breakpoint of 0.1 and the Disabled policy stands out from the rest of the configurations when handling small problems. The previous analysis suggests that the BioHEL system is more sensitive to changes in the configuration than GAssist. The best configuration found for GAssist (even though not significantly different from the rest) is $Auto_{1000}$ while for BioHEL is $Dis_{0.1}$.

4.5.2 Comparison with the best parameters per problem

In this section, we determine the best parameter settings per problem independently, and show how they compare to the fixed configurations obtained above for GAssist and BioHEL. First, we determine the problem-dependant parameter configurations for each system independently. Afterwards, we make the comparisons between the standard configurations and the best parameter settings found.

The procedure to determine the best parameters per problem in BioHEL and GAssist is the following:

1. Using the ensemble test accuracy recorded per problem for each 10-fold cross validation using 25 different seeds:
 - (a) The average accuracy per configuration is calculated as shown in Tables A.1 and A.4.

- (b) Independently per default class the best system parameter configuration is ruled out (the one that produced the highest test accuracy) and shown in Tables 4.9 and 4.11.
 - (c) The best configuration across all the default classes is shown in bold
2. A Kruskal-Wallis test is used to determine significant differences between the configurations within a particular default class.
 3. If significant differences are found the Wilcoxon pairwise test is used to compare the control method with the rest of the configurations.
 4. Frequency tables are constructed counting the cases in which particular scenarios presented significant differences with the rest (Tables 4.10 and 4.12).

In contrast to the previous section, the following analysis tries to find the best configuration per problem instead of finding the best across problems. This is the reason why the Friedman test is not suitable for this type of analysis as explain is Section 4.4.2.

4.5.2.1 Problem-dependent parameters for GAssist

In Table 4.9 we present the best results in terms of test accuracy for the 28 small problems for GAssist. We present the results for each default class policy, specifying the number of iterations that lead to that result. In this table we also emphasise the best results overall. Here we can observe that the number of iterations that produced the best results vary from problem to problem. However, in many cases the configuration that produces the best net results is not significantly better than the others within the scope of that particular problem. The configurations marked with a star are the ones that are in fact significantly different than others within the same default class. Here we can observe that for GAssist only the *pen* and *sat* problems present significant differences between using different configurations, which explains why a standard configuration could not be found in the previous section.

Moreover, Table 4.10 shows the frequency in which each of the number of iterations produce significantly results, taking also in account those configurations that where not significantly better that the control (please see full set of results in Appendix A). In this table we can observe that when a better configuration is ruled out this configuration corresponds to 1500 iterations. This could be due to the fact that the *pen* and *sat* problems are then larger problems in the small category. Larger problems require larger number of GA iterations as it will be shown in the analysis over large-scale datasets.

Even though problem dependent parameters could not be found for most problems in GAssist, we are going to use the configurations that produces the best results as a reference for later analysis in this chapter.

4.5.2.2 Problem-dependent parameters for BioHEL

Table 4.11 presents the best configurations found per problem for BioHEL, separated among the different default classes. In this case only the three default class policies accepted by the system are presented. Here we can observe that for some problems, the best coverage breakpoint differs among problems and default class policies. However, only for a subset of the problems the configuration that obtained the best test accuracy is significantly better than others. This subset is larger than the one found for GAssist and it also includes the *pen* and *sat* problems. For BioHEL, the policy that produces the best results more frequently is the default class policy, which is consistent with the results presented in Section 4.5.1.2. However, some problems still work better with other default class policies.

Table 4.9: Best test accuracy in GAssist using different default class policies. For each default class we show the number of iterations that produced the best result on the right and emphasise the best result overall default class policies. * appears only next to the configurations where that particular coverage break was significantly better than other configurations within that default class

Problem	Default class policy							
	Major		Minor		Disabled		Auto	
	Accuracy	Best Case	Accuracy	Best Case	Accuracy	Best Case	Accuracy	Best Case
bal	82.22±4.33	250	85.01±3.13	100	85.17±3.34	100	83.68±2.76	1500
bpa	65.81±8.18	500	68.15±7.46	250	68.44±6.90	1000	65.57±10.21	1000
bre	74.19±6.90	250	74.52±8.86	500	74.90±8.10	100	73.45±6.65	100
cmc	56.16±4.63	1000	54.66±3.10	100	55.47±4.23	100	56.09±3.87	1000
col	96.46±3.42	1500	97.01±2.42	500	97.28±2.27	500	97.00±3.06	250
cr-a	85.63±3.88	250	86.06±4.17	100	85.62±3.96	500	86.50±4.09	1000
gls	72.19±8.51	1500	69.02±9.87	1000	69.07±12.30	1000	70.80±10.70	1500
h-cl	81.47±4.10	250	83.16±5.26	100	83.11±6.12	100	83.17±5.42	100
hep	93.54±7.38	250	92.25±6.59	250	93.54±6.10	1000	92.29±7.23	100
h-h	96.60±3.93	1000	97.61±2.87	100	97.27±3.59	250	97.28±3.14	1500
h-s	80.37±8.56	100	83.33±7.86	100	83.33±6.36	1000	82.96±8.04	100
ion	93.16±4.32	100	94.00±4.19	1000	93.69±5.09	500	94.29±3.36	250
irs	95.33±5.49	100	96.00±4.66	250	96.00±4.66	1500	96.00±5.62	500
lab	100.00±0.00	100	100.00±0.00	100	100.00±0.00	100	100.00±0.00	100
lym	86.79±10.25	500	86.19±10.26	500	86.12±10.51	500	86.79±10.25	500
pen	89.16±0.98	1500 *	89.37±1.53	1500 *	89.06±0.94	1500 *	88.86±1.33	1500 *
pim	75.94±5.21	500	76.47±4.41	1000	75.95±4.94	500	75.81±5.00	500
prt	54.38±8.26	1500	53.48±6.91	1500	51.74±6.58	1000	51.95±4.30	500
sat	86.18±1.26	1500 *	84.02±1.22	1500 *	83.76±1.11	1000	84.37±1.25	1500 *
son	86.53±7.13	1000	80.34±6.00	250	77.95±9.15	1000	82.68±8.66	1000
thy	93.51±3.88	500	93.48±5.85	500	93.51±4.93	1500	93.51±5.86	250
vot	96.33±3.98	250	97.92±2.30	250	97.24±2.13	100	97.47±2.03	500
wav	83.06±2.47	1000	83.04±1.74	1500	83.28±2.09	1500	83.24±1.64	1500
wbcd	96.42±1.83	250	96.42±2.16	100	96.71±2.14	1000	96.85±2.11	100
wdbc	96.31±2.25	250	95.63±3.49	500	95.80±3.31	250	96.15±2.27	500
wine	98.82±2.48	100	97.68±3.00	100	97.16±3.98	1000	98.27±2.79	100
wpbc	82.70±8.68	250	79.64±8.09	1000	80.23±9.14	100	82.70±8.32	500
zoo	96.35±4.73	100	95.98±5.21	250	95.26±5.02	250	96.98±4.89	1500

Table 4.10: Frequency in which each one of the number of iterations in GAssist produced the best results using different default class policies

Default Class	Number of iterations for GAssist (frequency)				
	100	250	500	1000	1500
Majority	0	0	0	0	2
Minority	0	0	1	1	2
Disabled	0	0	0	0	1
Automatic	0	0	1	1	2
Total	0	0	2	2	7

Table 4.11: Best test accuracy in BioHEL using different default class policies. For each default class we show the coverage breakpoint that produced the best result on the right and * appears only next to the configurations where that particular coverage break was significantly better than other configurations within that default class

Problem	Default class policy					
	Major		Minor		Disabled	
	Average	Best Case	Average	Best Case	Average	Best Case
bal	83.34±3.82	0.05 *	87.66±3.35	0.1 *	87.53±3.77	0.1 *
bpa	66.18±7.56	0.01	71.03±6.47	0.1	69.62±6.85	0.05
bre	70.30±1.23	0.5	74.12±7.88	0.25	70.28±9.59	0.05
cmc	53.77±3.66	0.1 *	54.14±4.24	0.05	55.14±3.28	0.05 *
col	98.36±1.42	0.05	97.58±2.31	0.1	97.57±1.97	0.25
cr-a	85.49±4.79	0.1	86.06±3.95	0.1	87.07±5.33	0.1
glc	78.59±11.91	0.05	76.84±10.44	0.1	77.10±8.85	0.05
h-cl	82.23±6.26	0.05	80.54±4.95	0.1	80.54±3.49	0.01
hep	88.58±10.09	0.1	89.75±7.44	0.05	91.75±8.40	0.1
h-h	97.26±3.59	0.5	95.55±2.89	0.01	97.27±2.69	0.05
h-s	78.89±8.20	0.05	81.48±9.24	0.25	80.37±9.41	0.01
ion	94.35±4.15	0.1	93.44±3.37	0.5	92.60±3.35	0.1
irs	94.67±5.26	0.25	94.67±5.26	0.25	96.00±5.62	0.25
lab	98.57±4.52	0.5	96.33±7.77	0.05	98.57±4.52	0.5
lym	85.33±7.95	0.5	84.28±11.15	0.1	84.27±10.65	0.25
pen	98.93±0.42	0.01 *	98.97±0.26	0.01 *	98.96±0.29	0.01 *
pim	75.52±3.36	0.1	75.16±5.26	0.25	75.27±5.14	0.25
prt	45.79±7.77	0.01	48.91±6.34	0.05	49.83±7.80	0.05
sat	91.24±0.45	0.01 *	91.41±0.69	0.01 *	91.70±0.94	0.01 *
son	83.13±10.79	0.25 *	81.74±9.85	0.1	84.65±7.69	0.05
thy	93.98±3.02	0.01	94.85±5.59	0.25	95.41±6.43	0.05
vot	96.08±3.11	0.25	97.93±2.55	0.05	96.78±3.13	0.25
wav	84.68±1.95	0.05 *	85.02±1.95	0.01 *	85.80±1.67	0.05 *
wbcd	96.71±2.34	0.01	95.44±2.29	0.1	96.57±2.63	0.01
wdbc	96.66±1.76	0.01	96.35±3.74	0.25	96.51±3.43	0.1
wine	95.42±5.98	0.25	95.98±3.82	0.25	96.57±3.95	0.25
wpbc	77.36±12.27	0.1	82.55±8.04	0.05	78.94±7.07	0.05
zoo	92.25±7.34	0.5	94.98±5.31	0.25	94.98±5.31	0.1

Moreover, Table 4.12 shows the frequencies in which each coverage breakpoint produced significantly better results, also taking in account those configurations that were not dominated by the control. Here we can observe that the minority and disabled policy obtain better results with a coverage breakpoint of 0.01, while the the majority policy favours the coverage breakpoint 0.05 in most of the problems where differences were found. According to this table, the coverage breakpoints that produced the best results more frequently are 0.01 and 0.05, in contrast to the results obtained in the previous section. Our hypothesis is that when analysing the configurations per problem, the problems that require specific configurations are more likely the ones that need to develop a more complex solution and hence require using a more specific coverage breakpoint.

If we compare the best results found for BioHEL and GAssist, we can see that the best default class policies for each problem vary between systems. This is another indicator of the fact that this mechanism affects the systems in a different way. Moreover, there is no link between the coverage breakpoint used in BioHEL and the number of iterations used in GAssist for the same problem. However, it is noticeable that the largest problems favour either the usage of more specific coverage breakpoints or larger number of iterations.

Table 4.12: Frequency in which each one of the coverage breakpoints produced the best results in BioHEL using different default class policies

Def. Class	Coverage breakpoint for BioHEL (frequency)				
	0.5	0.25	0.1	0.05	0.01
Majority	0	2	3	6	4
Minority	0	0	1	2	4
Disabled	0	0	3	3	4
<i>Total</i>	0	2	7	11	12

4.5.2.3 Comparison with the standard configurations

Having analysed the best scenarios found for each problem, we are interested in comparing them with the standard configurations found in Section 4.5.1. In Table 4.13 we present the test accuracy of the best configurations found per problem according to Tables 4.9 and 4.11 and the results using the global default configuration for BioHEL ($Dis_{0.1}$) and GAssist ($Auto_{1000}$). This table also shows the percentage of accuracy loss considering the best solution and the results of the one-tailed Wilcoxon test to determine if there are significant differences between both configurations in each problem.

In this table we can see that the accuracy loss for both systems is not higher than 3% in 25 of the 28 problems presented, and not higher than 6% in the rest of the problems. Moreover, there are no significant differences between using a standard configuration and the best configuration in most of the problems. The only two problems that show significant differences are *pen* and *sat*, which belong to the largest problems of the small category. This was expected as this is consistent with the results shown in the previous section. At the end of the table we also show the p-values of the one-tailed Wilcoxon test to compare both configurations over the multiple datasets. For none of the systems there are significant differences, supporting the hypothesis that in general for small problems it is possible to use a standard configuration without degrading significantly the accuracy.

It can be argued that this comparison is strongly biased towards finding differences because the test accuracy of the best configuration per problem is the maximum of N trials, considering N the number of configurations tested, while the global configuration is the maximum of 1 trial. However, the fact that significant differences were not found makes stronger the assumption that the configurations are not different.

The results presented in this section show that there are no significant differences between the best configuration found per problem and the standard configuration found in Section 4.5.1. Therefore, we could state that the standard configurations found for each system can remain fixed when solving small problems.

4.5.3 GAssist vs. BioHEL

In this section we will compare BioHEL with GAssist over small problems in terms of learning quality, solution complexity and scalability of both systems.

To analyse the quality of the learning, the relationship between the training and test accuracies is shown in Table 4.14. The training and test accuracies help us determine whether one system learned better than the other (the test and training accuracies are both higher) or whether one system obtains worst results because it overfitted the data (the training accuracy is higher than

Table 4.13: Comparison of the best test accuracy found per problem (Best prob) with the test accuracy using the default configuration (Glob) for both systems. The W column shows a star if the Wilcoxon test determined significant differences between configurations ($\alpha = 0.05$) and the Acc % column show the % of accuracy loss with respect to the best solution found

Prob.	GAssist				BioHEL			
	Best Prob	Glob	W	Acc %	Best Prob	Glob	W	Acc %
bal	85.17±3.34	82.43±2.98		3.21	87.66±3.35	87.53±3.77		0.15
bpa	68.44±6.90	65.57±10.21		4.19	71.03±6.47	69.32±6.53		2.40
bre	74.90±8.10	72.78±5.36		2.84	74.12±7.88	69.57±8.31		6.13
cmc	56.16±4.63	56.09±3.87		0.12	55.14±3.28	53.50±4.21		2.96
col	97.28±2.27	96.46±3.21		0.84	98.36±1.42	97.57±1.51		0.80
cr-a	86.50±4.09	86.50±4.09		0.00	87.07±5.33	87.07±5.33		0.00
gls	72.19±8.51	68.46±11.89		5.17	78.59±11.91	76.45±10.07		2.73
h-c1	83.17±5.42	82.81±4.97		0.44	82.23±6.26	79.83±5.27		2.92
hep	93.54±7.38	92.25±6.59		1.38	91.75±8.40	91.75±8.40		0.00
h-h	97.61±2.87	96.59±3.93		1.04	97.27±2.69	95.26±3.67		2.07
h-s	83.33±7.86	81.85±7.08		1.78	81.48±9.24	79.26±7.03		2.73
ion	94.29±3.36	93.15±3.91		1.20	94.35±4.15	92.60±3.35		1.85
irs	96.00±4.66	96.00±5.62		0.00	96.00±5.62	94.00±4.92		2.08
lab	100.00±0.00	100.00±0.00		0.00	98.57±4.52	98.57±4.52		0.00
lym	86.79±10.25	86.79±10.25		0.00	85.33±7.95	80.76±12.52		5.35
pen	89.37±1.53	87.27±1.51	*	2.35	98.97±0.26	94.94±0.58	*	4.07
pim	76.47±4.41	75.81±5.00		0.86	75.52±3.36	74.74±2.77		1.02
prt	54.38±8.26	51.71±6.68		4.91	49.83±7.80	49.00±8.03		1.67
sat	86.18±1.26	83.74±0.94	*	2.83	91.70±0.94	88.14±1.50	*	3.88
son	86.53±7.13	82.68±8.66		4.46	84.65±7.69	84.56±8.85		0.11
thy	93.51±3.88	93.51±5.02		0.00	95.41±6.43	94.42±4.31		1.04
vot	97.92±2.30	97.46±2.30		0.47	97.93±2.55	96.08±4.11		1.89
wav	83.28±2.09	83.14±1.49		0.17	85.80±1.67	84.98±1.59		0.96
wbcd	96.85±2.11	95.71±2.03		1.19	96.71±2.34	95.43±1.75		1.32
wdbc	96.31±2.25	96.14±2.69		0.18	96.66±1.76	96.51±3.43		0.15
wine	98.82±2.48	96.10±3.76		2.75	96.57±3.95	94.80±5.77		1.83
wpbc	82.70±8.68	81.72±8.31		1.19	82.55±8.04	78.03±6.18		5.48
zoo	96.98±4.89	95.15±5.16		1.89	94.98±5.31	94.98±5.31		0.00
Global comparison p-value								
		0.2206				0.1948		

the other system, and the test accuracy is lower). In this table the first three rows correspond to the comparison among the standard parameter configurations using the different default class policies. This corresponds to 1000 iterations in case of GAssist and 0.1 coverage breakpoint in the case of BioHEL. The three following rows show the comparison of the best configurations per problem using the three different default class policies (as shown in Tables 4.9 and 4.11). The last two rows compare the overall standard and best configurations of the two systems. The symbols in table are the following: • BioHEL has better training and test accuracies, – BioHEL overfits the data, GAssist has better test accuracy, ◦ GAssist overfits the data, BioHEL has better test accuracy and * GAssist has better training and test accuracies.²

In this table we can see that, in most cases, BioHEL obtains a better training accuracy than GAssist. However, in half of these cases BioHEL overfits the data, and thus GAssist, overall, obtains better results. We can see that this happens less often when BioHEL uses the best coverage breakpoint found per problem. However, in general it seems that there are similar patterns for both the global and the best configurations in most problems. Also the dominance

²The analysis shown in these tables is purely qualitative and with the aim to support the results of the previous section. Hence it does not involve statistical tests.

Table 4.14: Comparison of the test accuracy and training accuracy of both systems. Global corresponds to the comparison of the standard configurations and Best corresponds to the comparison of the best configuration for each particular problem. •: BioHEL has better training and test accuracies, -: BioHEL overfits the data, GAssist has better test accuracy, ⊖: GAssist overfits the data, BioHEL has better test accuracy and *: GAssist has better training and test accuracies

Conf.		Problems																										Summary				
		bal	bpa	brc	cmc	col	cra	gls	h-cl	hop	h-h	h-s	ion	irs	lab	lym	pen	pum	prt	sat	son	thy	vot	wav	wbcd	wdbc	wine	wppc	zoo	.	.	
Global	Ma	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	11	0	10	7
	Mi	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	14	1	10	3
	Di	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	12	0	13	3
Best	Ma	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	12	1	8	7
	Mi	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	14	0	10	4
	Di	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	13	1	11	3
	G	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	11	0	14	3
	B	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	12	0	11	5

of one system above the other seems to depend on the characteristics of the system, rather than the usage of specific parameters.

To continue with this line of analysis, Table 4.15 shows a similar comparison, but between the test accuracy and the number of rules. The goal is to obtain solutions that are accurate, but also as general and compact as possible. Therefore, if the solution has less rules and higher test accuracy then it is better than the other. The symbology to read the table is the following: • means BioHEL has better test accuracy and less number of rules, ◦ means BioHEL has better test accuracy but higher number of rules, ◡ means GAssist has better test accuracy but higher number of rules, and ★ means GAssist has better test accuracy and less number of rules. In this table it is noticeable that BioHEL usually has larger number of rules than GAssist, but in many cases GAssist also has better test accuracy. Considering these two aspects there seems to be a dominance of the GAssist system over BioHEL. Also this table seems to be the inverse of Table 4.14 in most cases. This was expected because when the training accuracy is higher, it is usually because the system generated a more complex set of rules. Also, GAssist is a Pittsburgh LCS, hence it has an overview of the rule set generation and penalises large rule sets. On the other hand, BioHEL generates each rule independently, without a global supervision of the constructed solution.

As it was already mentioned, the previous tables suggest that there is a connection between the characteristics of the problems and the obtained results. In Figure 4.2 all the problems are ordered by the number of classes in a radial graph (starting anticlockwise from the numbered axis) and plotted the relative test accuracy of BioHEL with respect to GAssist. This means that when the value in the figure is below 1, GAssist performed better, and when the value is higher than 1, BioHEL performed better. In this figure we can observe that GAssist behaves better when the system has less classes, while BioHEL behaves better when the system has more classes, with the exception of some problems. It is also interesting to notice that in the problem with 21 classes none of the systems presented good results. In BioHEL this problem is particularly difficult, because when the system tries to adjust the coverage breakpoint according to the class distribution the system loses specificity pressure. On the other hand, it was expected that this problem was very difficult for GAssist. This particular weakness of GAssist when handling a large number of classes was already reported in [Bacardit and Butz, 2007].

Furthermore, we also ordered the problems according to the percentage of continuous attributes as shown in Figure 4.3. We can see here that GAssist works better with problems that have discrete attributes and BioHEL works better in problems that have more continuous

Table 4.15: Comparison of the test accuracy and number of rules of both systems. Global corresponds to the comparison of the standard configurations and Best corresponds to the comparison of the best configuration for each particular problem. ●: BioHEL has better test accuracy and less number of rules, ○: BioHEL has better test accuracy but higher number of rules, —: GAssist has better test accuracy but higher number of rules, and ★: GAssist has better test accuracy and less number of rules

Conf.		Problems																										Summary					
		bal	bpa	bre	cmc	col	cr-a	glis	h-cl	hep	h-h	h-s	ion	irs	lab	lym	pen	pim	prt	sat	son	thy	vot	wav	wbcd	wdbc	wine	wpbc	zoo	●	○	—	★
Global	Ma	○	○	—	—	●	○	○	★	—	★	★	○	★	—	★	○	●	★	○	★	★	★	○	★	○	—	★	—	2	9	6	11
	Mi	○	○	★	—	○	○	○	★	★	—	★	★	—	—	○	○	★	★	○	○	○	○	○	★	○	—	○	●	1	14	4	9
	Di	○	○	★	★	○	○	○	★	★	★	★	★	★	—	★	○	★	★	○	○	○	★	○	★	○	—	★	●	1	11	2	14
Best	Ma	○	○	—	—	●	★	○	○	★	●	★	○	—	—	—	○	—	★	○	★	○	★	○	○	○	—	★	—	2	11	8	7
	Mi	○	○	★	★	●	○	○	○	★	○	●	—	—	—	★	○	★	○	★	○	●	○	○	★	○	—	○	—	3	11	4	10
	Di	○	○	★	★	○	○	○	★	★	○	★	★	★	—	★	○	★	★	○	★	○	○	○	○	○	—	★	—	0	14	3	11
	G	○	○	★	—	○	○	○	★	★	★	★	★	★	—	★	○	★	★	○	○	○	★	○	★	○	—	★	—	0	11	4	13
	B	○	○	★	★	○	○	○	★	★	★	★	○	★	—	—	○	—	★	○	★	○	○	○	★	○	—	★	—	0	12	5	11

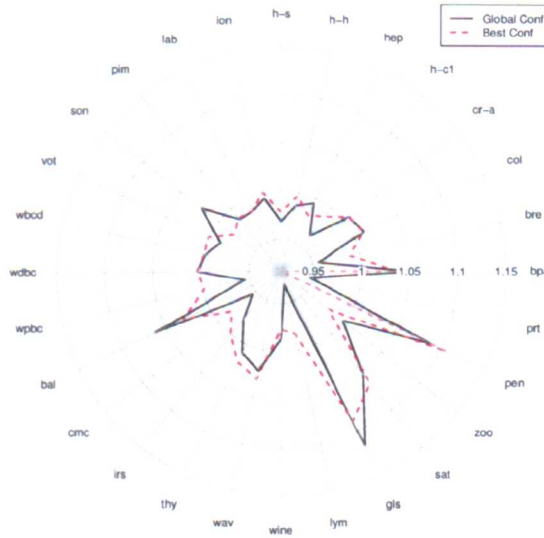


Figure 4.2: Relative test accuracy of BioHEL according to GAssist in the small problems ordered by the number of classes, starting anticlockwise from the numbered axis. Data points below one represent that GAssist performs better and data points over one represent the opposite.

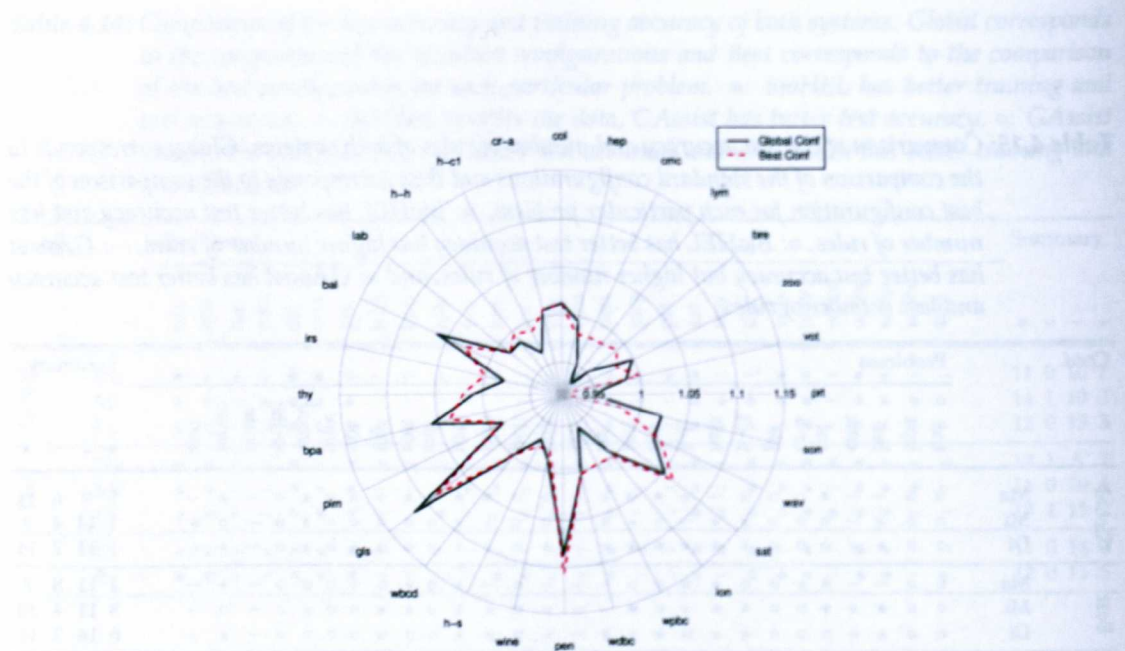


Figure 4.3: Relative test accuracy of BioHEL according to GAssist in the small problems ordered by the percentage of continuous attributes, starting anticlockwise from the numbered axis. Data points below one represent that GAssist performs better and data points over one represent the opposite.

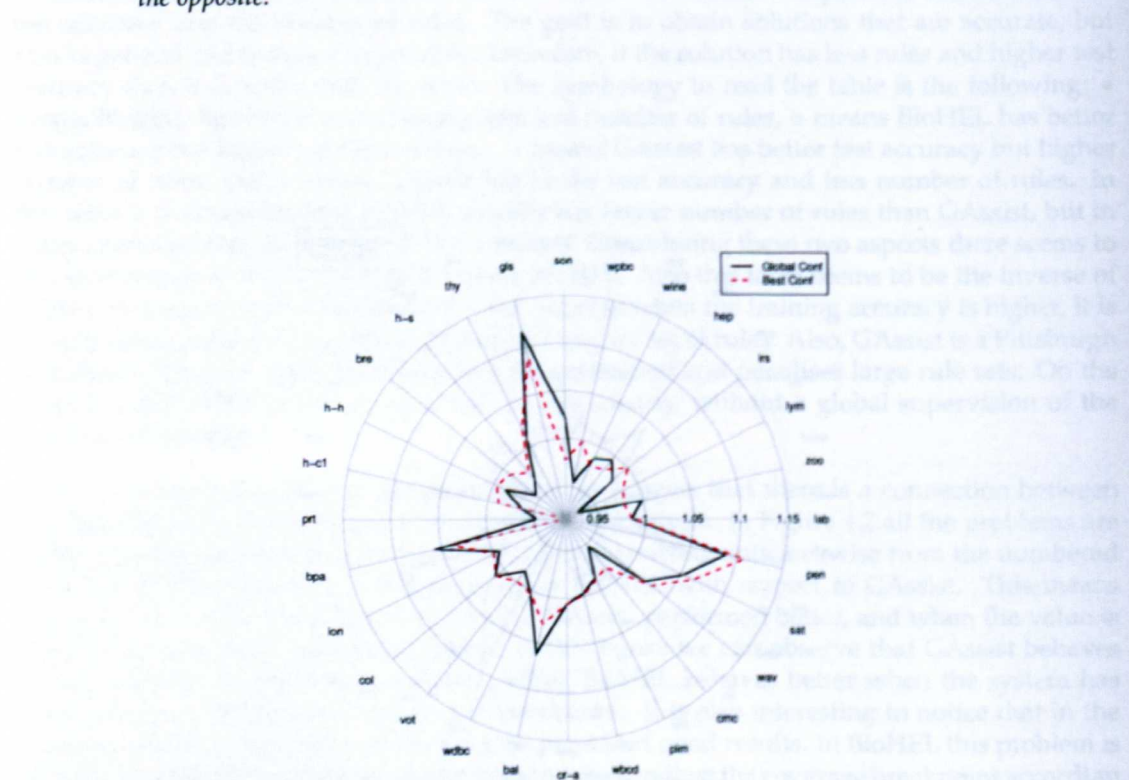


Figure 4.4: Relative test accuracy of BioHEL according to GAssist in the small problems ordered by the number of instances, starting anticlockwise from the numbered axis. Data points below one represent that GAssist performs better and data points over one represent the opposite.

attributes (with the exception of some problems). This difference is due to the different representations used by the two systems, in the case of BioHEL hyper-rectangles and in the case of GAssist the ADI representation. Moreover, if we order the problems by the number of instances (Figure 4.4) we can see that GAssist performs better in the small problems (with the exception of *gls* and *thy*, which have large number of classes). On the other hand, BioHEL seems to dominate in the large ones. This was expected since BioHEL was developed to handle larger domains than GAssist.

As it was explained before, BioHEL is based on GAssist. However, this system changed few mechanisms along with the learning paradigm in order to cope with larger domains. In this sense, BioHEL is known to be faster than GAssist. In Table 4.16 we present the speedup of BioHEL over GAssist in all the cases analysed before. This table shows that, in very few cases, GAssist performs faster than BioHEL. On the other hand, BioHEL achieves speedups up to 243X. This is because when the system needs to generate a small number of rules the learning iterations are less in BioHEL than in GAssist. It is also worth noticing that the speedups change drastically among configurations. The majority and the disabled policies tend to help BioHEL achieve larger speedups. This could be because the majority policy reduces the amount of rules the system needs to generate. Also, even though the disabled policy does not reduce the amount of rules, the problem the system learns is easier.

Table 4.16: Speedup of the BioHEL system over GAssist for the different configurations. The configurations where GAssist is faster are emphasised

Problem	Global Conf.			Best Conf.			Global	Best
	Major	Minor	Dis	Major	Minor	Dis		
bal	5.89	3.34	3.77	0.87	0.31	1.31	4.45	3.42
bpa	7.59	5.00	4.32	1.97	1.13	1.01	7.29	6.96
bre	4.20	1.11	0.95	2.55	0.92	0.36	1.65	0.40
cmc	8.08	5.65	5.23	8.08	2.34	0.17	7.13	3.56
col	14.72	13.25	15.56	22.65	25.80	9.64	19.04	6.93
cr-a	3.41	4.18	3.74	0.78	0.39	1.31	5.67	1.51
gls	6.83	5.11	5.73	9.72	10.23	4.84	5.85	10.63
h-cl	5.97	5.13	5.04	1.33	0.47	0.85	6.82	0.79
hep	10.83	7.32	9.71	2.76	1.70	3.41	11.78	15.99
h-h	11.16	9.51	10.01	20.06	9.99	2.18	14.14	5.53
h-s	14.02	12.74	11.91	1.27	1.93	5.73	16.96	2.41
ion	39.30	67.28	65.49	4.27	11.19	35.41	88.33	243.62
irs	7.43	7.43	8.24	1.18	1.17	4.21	10.36	17.69
lab	14.52	10.69	22.85	1.66	1.46	2.78	24.81	22.96
lym	5.45	4.57	5.23	3.18	2.44	2.90	5.30	1.58
pen	25.86	26.05	27.76	16.18	16.20	17.47	26.65	26.37
pim	13.49	5.55	4.80	6.34	14.14	6.89	8.51	72.00
prt	1.32	0.94	0.90	0.74	1.34	0.60	0.79	1.60
sat	39.48	40.33	49.91	13.25	17.46	9.45	57.59	54.01
son	81.60	83.54	114.88	91.12	85.79	21.95	104.38	162.61
thy	11.46	9.22	9.03	4.38	6.10	10.64	12.21	26.45
vot	1.60	1.28	1.12	0.49	0.33	0.72	1.48	1.97
wav	24.09	23.08	29.28	12.19	6.10	22.41	34.18	52.42
wbcd	13.20	15.61	16.75	2.00	1.41	2.13	23.65	0.67
wdbc	46.09	50.75	51.94	7.03	29.86	18.36	72.16	23.50
wine	29.20	25.56	32.48	3.53	3.13	33.44	38.11	46.57
wpbc	35.13	30.14	37.51	9.25	26.69	9.75	39.68	8.63
zoo	2.01	1.71	2.08	0.20	0.21	0.52	2.22	3.29

Table 4.17: Test accuracy in BioHEL using different coverage breakpoints over large problems

Problem	Coverage breakpoint (% Test accuracy)				
	0.001	0.005	0.01	0.05	0.1
Adu	86.09±0.39	86.33±0.46	86.14±0.41	83.95±0.83	83.08±0.59
c-4	80.94±0.46	78.02±0.33	76.33±0.31	69.56±0.40	65.83±0.02
SS	71.19±1.13	70.33±1.01	68.65±1.34	61.26±1.20	47.70±1.22
PMX	100.00±0.00	92.58±1.22	85.92±1.94	50.00±0.00	50.00±0.00
kdd	99.95±0.01	99.91±0.01	99.92±0.02	99.87±0.01	99.76±0.03
SA	79.30±0.33	78.79±0.34	78.44±0.33	76.96±0.34	75.84±0.51
CN	80.59±0.46	79.78±0.39	79.22±0.37	77.21±0.52	75.83±0.57

Table 4.18: Test accuracy in GAssist using different number of iterations over large problems

Problem	Number of Iterations (% Test accuracy)			
	1000	2500	5000	10000
Adu	85.87±0.46	85.94±0.43	86.07±0.36	86.08±0.49
c-4	76.17±0.59	77.95±0.44	78.98±0.24	79.77±0.37
SS	59.82±1.88	61.75±1.38	62.58±1.38	62.98±1.53
PMX	67.36±1.58	74.69±1.88	82.68±1.98	87.50±2.17
kdd	99.05±0.11	99.17±0.09	99.21±0.14	99.25±0.12
SA	76.16±0.45	76.51±0.27	—	—
CN	77.89±0.41	78.05±0.45	—	—

When comparing the systems to each other, GAssist shows superiority over BioHEL in small problems, since it generates more compact rule sets and obtains a higher accuracy. On the other hand, BioHEL generates larger rule sets and overfits the data. Regarding the domains of competence of each system, GAssist seems to behave better in problems with either less classes and/or less instances, while BioHEL obtains good results in problems with opposite characteristics. Also BioHEL seems to perform better in problems with continuous attributes, while GAssist presents better results with discrete problems.

4.5.4 Parameter-sensitivity analysis over large scale datasets

In this section we perform a similar but less extensive analysis of parameter sensitivity over large problems. For these problems we did not perform the analysis over all the default class policies, but over the majority policy only. The usage of this policy reduces the search space (by reducing the amount of instances that should be classified by the generated rules), which is more convenient for big problems. Even though in some cases, especially in small problems, this might make the problem more difficult, in large domains it is beneficial to pick a default class [Bacardit, 2004].

Tables 4.17 and 4.18 show the test accuracy over the big problems in BioHEL and GAssist,³ respectively. In BioHEL (Table 4.17) we can see that the best configuration is 0.001 for all problems with the exception of *Adu*, where the best coverage breakpoint is 0.005. Similar results were found in GAssist, where the usage of 10000 iterations produces the highest test accuracy. We can observe that these configurations are the ones that produce the more specific solutions. This in general can cause data overfitting. However, BioHEL uses ensembles to perform the

³In the case of GAssist the results with some configurations are missing, since the runs for these configurations took more than 10 days each, which is one of the constraints of our computational framework.

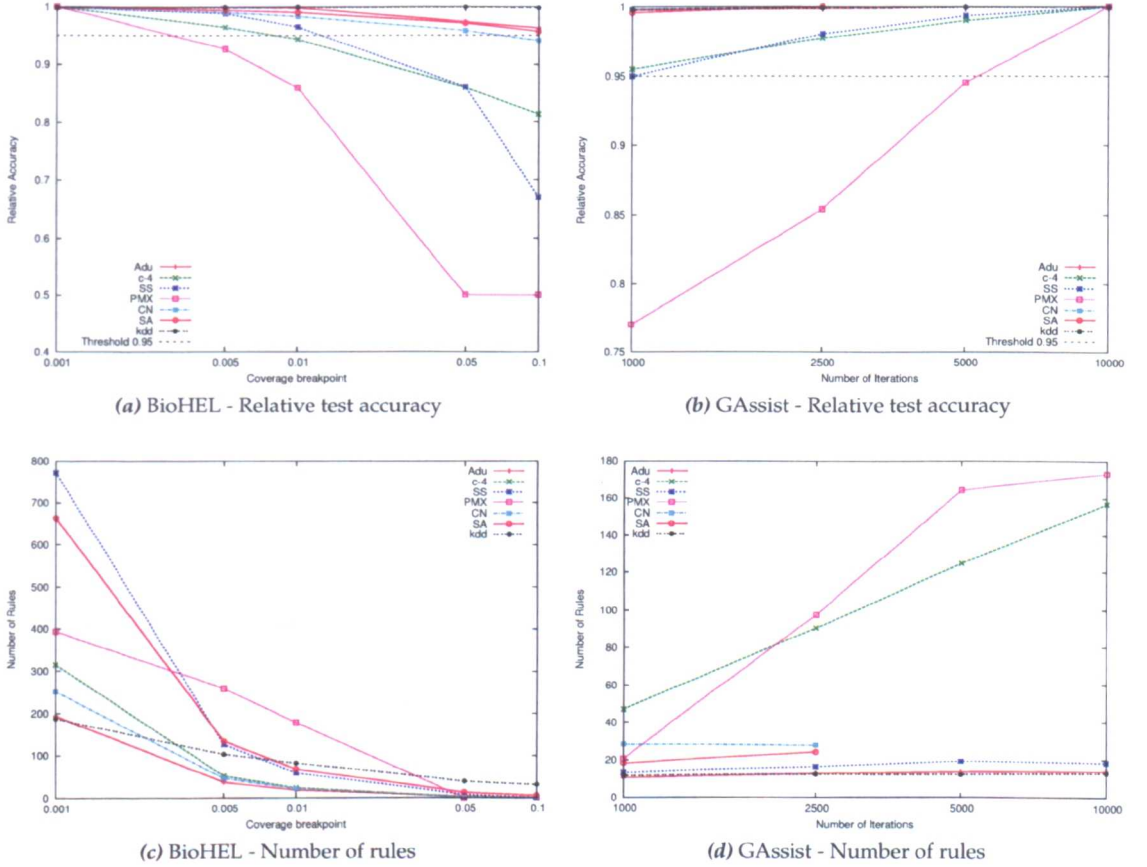


Figure 4.5: Relative test accuracy (considering best value observed) and number of rules obtained with different parameter configurations in large datasets

final classification, which benefit from having diverse specific models that together produce a more robust prediction.

Figure 4.5 shows how the relative accuracy⁴ and number of rules vary according to the coverage breakpoint and the number of iterations. Figures 4.5a and 4.5b show how the BioHEL system is more sensitive to changes in the parameters than the GAssist system. We can see in these figures that even when BioHEL obtains better results than GAssist for all problems, when using a non-adequate coverage breakpoint, the accuracy drops drastically. Moreover, in Figures 4.5c and 4.5d it is noticeable that while the coverage breakpoint decreases (BioHEL) or the number of iterations increase (GAssist), the number of rules also increase. This is because these parameters increase the specificity pressure of the system, forcing the system to develop more fine-grained solutions. This is even more noticeable in BioHEL, where the number of rules generated increases in an exponential manner. Moreover, the number of rules in BioHEL is very large compared to the number of rules generated by GAssist. This is because the learning of each rule in BioHEL is independent from each other, while in GAssist it is a parallel process, in which the size of the rule set is constraint by the algorithm via fitness penalties.

Moreover, in Figures 4.5a and 4.5b a horizontal line indicates when the relative accuracy drops below 0.95. In real life problems sometimes the best solution is not the one that has the highest accuracy, but the one that is close enough to the best one, but at the same time is compact and simple. In these figures we can see that it is possible to apply more pressure over the systems by increasing the coverage breakpoint or decreasing the number of iterations. The increase in

⁴The accuracy of the scenario divided by the largest accuracy obtained.

Table 4.19: Test accuracy in BioHEL using different number of windows

Problem	Number of Windows (% Test accuracy)				
	10	20	50	100	200
Adu	86.09±0.39	86.41±0.35	86.34±0.38	86.16±0.37	85.83±0.32
c-4	80.94±0.46	80.77±0.41	79.44±0.35	78.46±0.38	77.71±0.49
SS	71.19±1.13	70.83±1.00	69.78±1.14	68.86±0.94	68.04±1.19
PMX	100.00±0.00	100.00±0.00	100.00±0.00	92.93±1.54	86.67±0.48
kdd	99.95±0.01	99.94±0.01	99.94±0.01	99.93±0.02	99.91±0.02
SA	79.30±0.33	79.17±0.31	79.00±0.34	78.79±0.31	78.62±0.36
CN	80.59±0.46	80.49±0.36	80.26±0.41	79.96±0.41	79.64±0.44

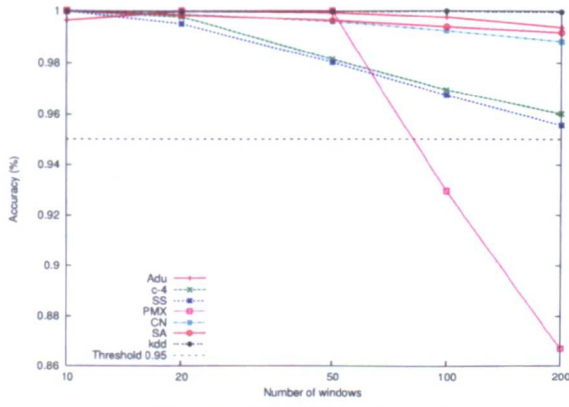
Table 4.20: Test accuracy in GAssist using different number of windows

Problem		Number of Windows (% Test accuracy)				
		10	20	50	100	200
10000 iterations	Adu	86.08±0.49	85.95±0.39	85.83±0.48	85.66±0.59	85.63±0.49
	c-4	79.77±0.37	78.96±0.46	77.47±0.40	76.29±0.42	74.96±0.53
	SS	62.98±1.53	62.50±0.98	61.65±0.92	60.66±1.23	59.00±1.41
	PMX	87.50±2.17	83.54±3.39	79.43±3.72	76.05±2.01	73.15±2.69
	kdd	99.25±0.12	99.22±0.02	99.21±0.01	99.20±0.02	99.18±0.02
	SA	–	–	76.76±0.26	76.52±0.34	76.49±0.32
	CN	–	–	77.70±0.40	77.50±0.58	77.44±0.54
2500 iterations	SA	76.51±0.27	76.47±0.34	76.30±0.33	76.31±0.27	76.13±0.28
	CN	78.05±0.45	77.72±0.55	77.67±0.26	77.38±0.56	77.25±0.55

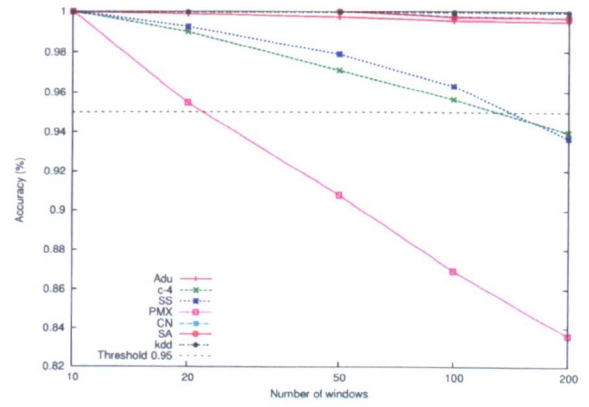
the pressure helps obtain a solution that is also good but is much more compact. However, this is not applicable to all problems. As we can see, in synthetic problems, such as PMX (parity multiplexer problem), the accuracy drops much more drastically from one configuration to the other.

In this section a comparison between the best configuration found and the standard configuration is not performed, since in all of the problems with the exception of one, both values are the same. In the following sections we are going to present the analysis of another shared mechanism between the systems: the ILAS windowing scheme.

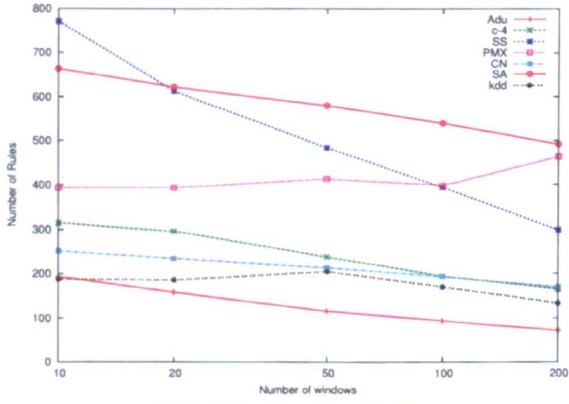
For BioHEL, the coverage breakpoint that produces the best results for large problems is 0.001, while for GAssist the number of iterations that produced the best results was 10000 iterations. Moreover, in the large problems BioHEL obtains better results than GAssist in terms of test accuracy, but also it is much more sensitive to changes in the configuration. It generates larger rule sets that grow exponentially when the coverage breakpoint decreases. Also it was shown that the coverage breakpoint as well as the number of iterations are parameters that can be used to regulate the generalisation pressure of the system. This is useful to find more compact rule sets without severely degrading the accuracy.



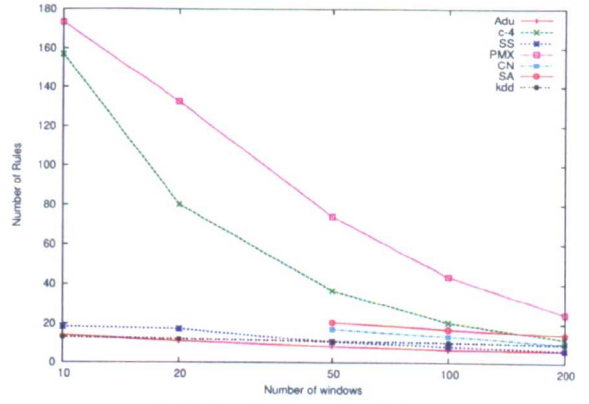
(a) BioHEL - Relative test accuracy



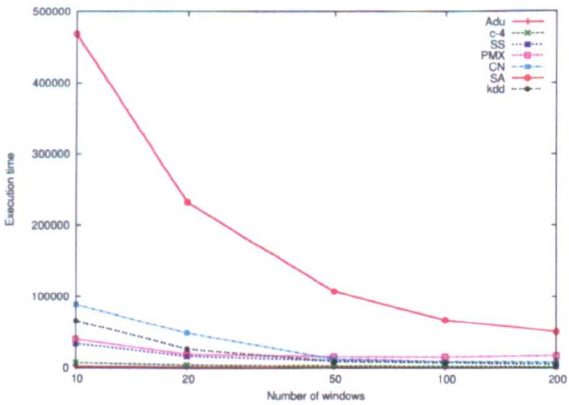
(b) GAssist - Relative test accuracy



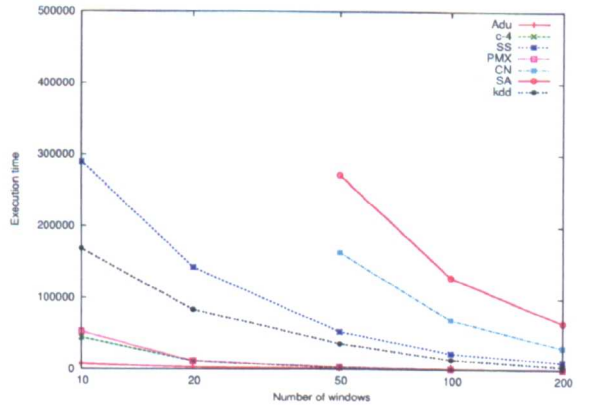
(c) BioHEL - Number of rules



(d) GAssist - Number of rules



(e) BioHEL - Execution time



(f) GAssist - Execution time

Figure 4.6: Relative test accuracy (considering best value observed), number of rules and execution time obtained with different window sizes in large datasets

4.5.5 Analysis of the ILAS windowing scheme

Tables 4.19 and 4.20 show the results of varying the number of ILAS windows in BioHEL and GAssist, respectively. For the analysis of the ILAS windowing scheme we considered only a coverage breakpoint value of 0.001 for BioHEL and 10000 GA iterations for GAssist, which were the configurations that produced the best results in previous sections. Also for the SA and CN problems in GAssist, we show the behaviour using 2500 iterations, since we do not have data for 10000 iterations. In these tables we can see that, while the number of windows increases, the accuracy decreases. Nevertheless, the usage of a higher number of windows in BioHEL is beneficial in the *Adu* problem, since it helps the system to generalise. In the case of BioHEL, we can observe that more generalisation pressure can be achieved by either using a more strict coverage breakpoint or using a larger number of windows, as it was shown for the *Adu* problem. The same behaviour is not observed in GAssist, where all the problems lose accuracy with the increase of the number of windows.

In Figure 4.6 we show the changes in the relative accuracy, the number of rules and the total execution time when using different number of windows. In Figures 4.6a and 4.6b we can observe that the relative accuracy decreases with the number of windows in both systems in a similar manner. Moreover, the accuracy decreases drastically in synthetic problems such as PMX, behaviour that can be explained by ILAS's theoretical models (see Section 2.4.1.3). This figure also shows a line where the relative accuracy drops below 0.95. It is noticeable that for all the problems except PMX it is possible to use a higher number of windows, without compromising the accuracy of the system severely. Figures 4.6c and 4.6d show the effect of the number of windows over the number of rules. The changes are more dramatic in GAssist than in BioHEL, where the number of rules decrease exponentially with the number of windows. This could be due to the fact that from the start GAssist generates much smaller rule sets than BioHEL. Also, BioHEL does not reduce the rule set size drastically. On the contrary, for the synthetic problem PMX, a high number of windows produces larger rule sets. It might happen that some of the strata are not good representations of the whole training set and this prevents the system from learning, either a correct rule (with 100% accuracy) or a general enough rule. The latter case is the main reason for the generation of large rule sets.

Moreover, Figures 4.6e and 4.6f shows the decrease of the execution time with respect to the number of windows. This was expected since the main goal of this technique is to decrease the computational time during the match process. However, we can see in these figures that the execution times for BioHEL are considerably smaller than the execution times for GAssist, although the latter generates smaller rule sets.

In this section we showed how the windowing mechanism can help reduce the execution time of both systems without degrading the accuracy severely. Moreover, the windowing can be used as a tool to introduce generalisation pressure, in order to obtain solutions that are good enough, but are more compact in terms of rules. Nevertheless, the impact of this mechanism in terms the compactness of the rule sets differ from GAssist and BioHEL. While in GAssist the number of rules decrease drastically with the number of windows in BioHEL the changes are not so noticeable. On the other hand, BioHEL generates larger rule sets, but the execution times over large problems are much smaller than GAssist. This fundamentals the hypothesis that the BioHEL system is more suitable for larger domains.

4.5.6 Comparison against other ML techniques

In this section we compare the performance of GAssist and BioHEL with other ML algorithms such as: C4.5, PART, IBk, Naïve Bayes and the SMO implementation of SVM with both polynomial and gaussian kernels.

Table 4.21: Comparison of GAssist and BioHEL with other ML algorithms in terms of test accuracy. The cells in bold emphasise the best result obtained for each problem

Prob.	GAssist	BioHEL	C4.5	NB	SVM	SVM _r	PART	IB5
bal	82.43±2.98	87.53±3.77	78.10±3.46	90.11±1.76	88.02±2.62	90.89±1.62	82.09±4.19	86.09±2.36
bpa	65.57±10.21	69.32±6.53	66.36±6.86	55.08±8.26	57.99±1.17	57.99±1.17	65.52±8.58	57.72±5.44
bpe	72.78±5.36	69.57±8.31	74.55±7.07	72.81±10.18	70.99±8.28	72.39±7.37	65.45±8.56	73.05±4.24
cmc	56.09±3.87	53.50±4.21	51.14±3.63	50.73±4.28	49.37±4.40	46.23±4.06	50.65±4.04	47.73±4.34
col	96.46±3.21	97.57±1.51	85.33±4.12	78.73±7.79	83.42±5.05	75.42±9.03	84.79±3.67	81.22±4.69
cr-a	86.50±4.09	87.07±5.33	84.90±4.65	77.68±4.01	85.05±3.94	85.63±3.86	84.17±5.22	86.34±4.73
gls	68.46±11.89	76.45±10.07	68.36±11.09	50.39±10.72	57.43±12.39	58.98±13.95	69.65±11.71	67.74±12.55
h-cl	82.81±4.97	79.83±5.27	76.85±6.02	83.51±7.74	83.17±5.70	82.77±6.07	78.61±4.36	82.15±5.48
hep	92.25±6.59	91.75±8.40	77.96±11.61	84.50±6.31	85.79±5.80	84.46±5.61	80.62±8.85	85.17±7.50
h-h	96.59±3.93	95.26±3.67	80.66±7.15	85.39±7.82	84.35±8.37	78.90±6.73	82.02±5.73	80.61±5.65
h-s	81.85±7.08	79.26±7.03	80.74±7.16	84.44±6.00	84.44±7.37	81.48±6.76	77.41±7.90	80.00±5.00
ion	93.15±3.91	92.60±3.35	89.96±4.51	82.94±7.61	87.77±3.74	94.84±4.74	91.12±4.24	84.30±5.74
irs	96.00±5.62	94.00±4.92	93.33±5.44	94.67±5.26	96.00±4.66	96.67±4.71	92.67±4.92	94.67±6.89
lab	100.00±0.00	98.57±4.52	84.05±13.14	95.71±13.55	92.29±10.69	95.48±7.31	83.19±9.94	88.86±15.68
lym	86.79±10.25	80.76±12.52	78.21±15.08	81.96±8.73	85.65±12.12	90.45±0.25	75.33±14.32	83.35±10.82
pen	87.27±1.51	94.94±0.58	96.62±0.30	85.76±1.03	97.94±0.39	99.45±0.25	96.89±0.48	99.22±0.25
pim	75.81±5.00	74.74±2.77	74.87±4.30	74.75±4.59	76.98±4.38	77.64±4.31	74.91±5.26	74.36±4.59
prt	51.71±6.68	49.00±8.03	41.35±6.68	50.71±8.56	47.36±8.00	44.22±6.48	40.64±6.40	47.98±3.97
sat	82.68±8.66	88.14±1.50	86.68±1.22	79.63±1.47	86.88±1.39	89.35±1.23	86.54±1.12	90.91±1.40
son	93.51±5.02	94.42±4.31	92.08±5.43	97.21±3.93	76.38±8.25	85.13±6.04	71.45±8.88	82.31±8.37
thy	97.46±2.30	96.08±4.11	95.85±3.92	90.11±4.34	90.26±5.53	88.90±6.53	94.91±5.04	93.96±4.37
vot	83.14±1.49	84.98±1.59	74.96±2.47	79.86±1.63	95.85±2.87	96.55±3.30	96.08±3.81	93.58±3.69
wav	95.71±2.03	95.43±1.75	94.42±2.98	96.00±2.50	86.70±1.70	86.70±1.67	77.96±2.10	79.80±1.79
wbcd	96.14±2.69	96.51±3.43	93.49±4.11	92.96±3.35	96.71±1.66	96.85±2.01	95.14±3.18	96.85±1.47
wdbc	96.10±3.76	94.80±5.77	91.43±7.76	96.60±4.04	97.73±2.32	98.26±1.82	94.39±3.36	96.86±2.79
wine	96.10±3.76	94.80±5.77	91.43±7.76	96.60±4.04	98.82±3.72	98.30±3.90	90.88±6.83	95.39±4.62
wpbc	81.72±8.31	78.03±6.18	66.81±7.88	67.85±8.59	77.51±6.56	78.08±6.95	69.77±8.75	75.07±9.70
zoo	95.15±5.16	94.98±5.31	92.35±6.23	93.98±7.02	95.98±5.21	93.16±6.40	92.35±6.23	91.33±5.31
Adu	86.08±0.49	86.09±0.39	85.99±0.42	83.24±0.44	84.93±0.36	83.66±0.55	85.64±0.54	82.62±0.56
c-4	79.77±0.37	80.94±0.46	80.89±0.57	72.11±0.19	75.85±0.35	83.95±0.27	79.22±0.49	81.03±0.17
PMX	87.50±2.17	100.00±0.00	76.99±14.35	48.91±0.56	49.59±0.70	76.27±0.16	59.89±16.93	87.58±0.19
SS	62.98±1.53	71.19±1.13	55.12±0.77	67.73±0.76	73.20±1.08	71.75±1.09	58.92±0.77	58.25±2.52
kdd	99.25±0.12	99.95±0.01	99.96±0.01	92.27±0.14	99.93±0.01	99.93±0.01	99.97±0.01	99.93±0.01
SA	76.76±0.26	79.30±0.33	70.96±0.27	75.12±0.40	-	-	71.88±0.00	75.53±0.33
CN	77.70±0.40	80.59±0.46	73.14±0.43	75.68±0.60	80.18±0.37	83.34±0.49	74.79±0.36	78.31±0.66

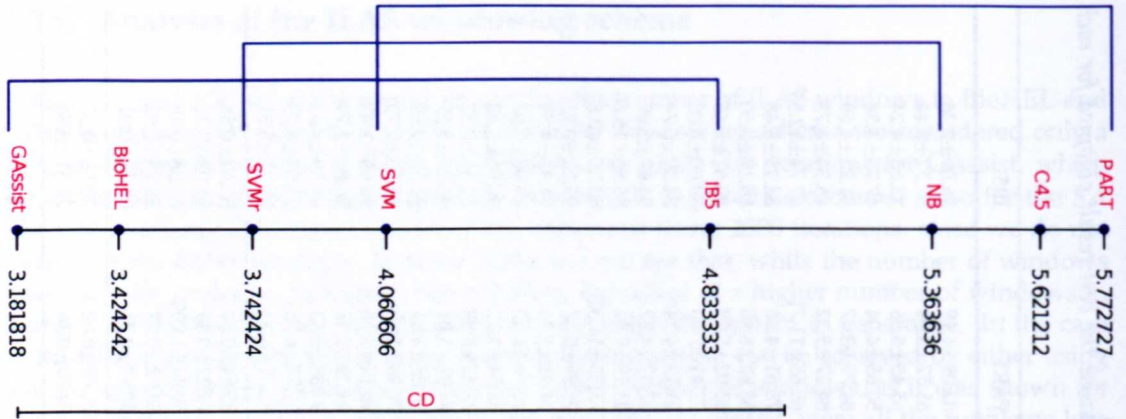


Figure 4.7: Clustering of the different ML algorithms using Nemenyi critical distance

Table 4.21 shows the results of the comparison of GAssist and BioHEL with the mentioned algorithms. This table only shows the results using the standard configurations found for BioHEL and GAssist in Section 4.5.1, in order to provide a fair comparison with the other methods.⁵ In this table we can see that GAssist and BioHEL are among the best algorithms. Only SVM with gaussian kernels (SVNr) seems to achieve good solutions as frequently as our algorithms. It is worth noticing that for the SA problem some of the ML algorithms did not manage to learn the problem due to the lack of memory and impractical execution times. Moreover, since we do not have results for the SA and CN problems using the standard configuration of GAssist (10000 iterations and 10 windows), we included the results using 50 windows instead.

When performing the Friedman test to determine if there were significant differences we obtained a p-value of $9.1e^{-7}$. In Figure 4.7 we can visualise graphically the results of the post-hoc test to compare all algorithms using the Nemenyi critical distance. In this case the critical distance is equal to 1.827. It is worth noticing that for this statistical test the SA problem was not considered since we did not have results for all the algorithms. In this test we can observe that GAssist and BioHEL place first and second, respectively. Being in the cluster of the best algorithms, along with SVM (with both polynomial and gaussian kernels) and IB5, there are only significant differences between them and NaïveBayes, C4.5 and PART.

Moreover, when comparing the tested ML methods in terms of execution time, we can obtain interesting insights. Table 4.22 shows the execution times over the large problems and the respective rankings. Moreover, Figure 4.8 shows the information from the previous table plotted independently per problem. In general, we can observe that the algorithms present consistent rankings among each other and that SVMr and GAssist have the highest execution times. On the other hand BioHEL, SVM and IB5 have similar execution times, except in the two largest problems, where the polynomial kernels (SVM) scale much worse.⁶ Only in the *kddcup* problem SVMr takes half the time than BioHEL and SVM takes only 2% of the time. For the rest of the problems BioHEL seems to be faster. Moreover, C4.5, NaïveBayes and PART have very small execution times compared to the other algorithms, but these algorithms do not give good results in terms of accuracy, as it was shown before. Also, PART scales badly in problems with large amount of attributes such as SS, SA and CN.

It can be arguable that no stratification techniques are applied to SVM and this is the reason why these algorithms are not performing faster. However, the ILAS Windowing technique is a built-in technique within BioHEL and GAssist, and the original implementation of SVM

⁵For these algorithms an analysis to the determine the best standard configuration for all problem was performed in a similar fashion as in Sections 4.5.1.1 and 4.5.1.2.

⁶For the SA problem the execution time for SVM took more time than the allowed time of our computational framework

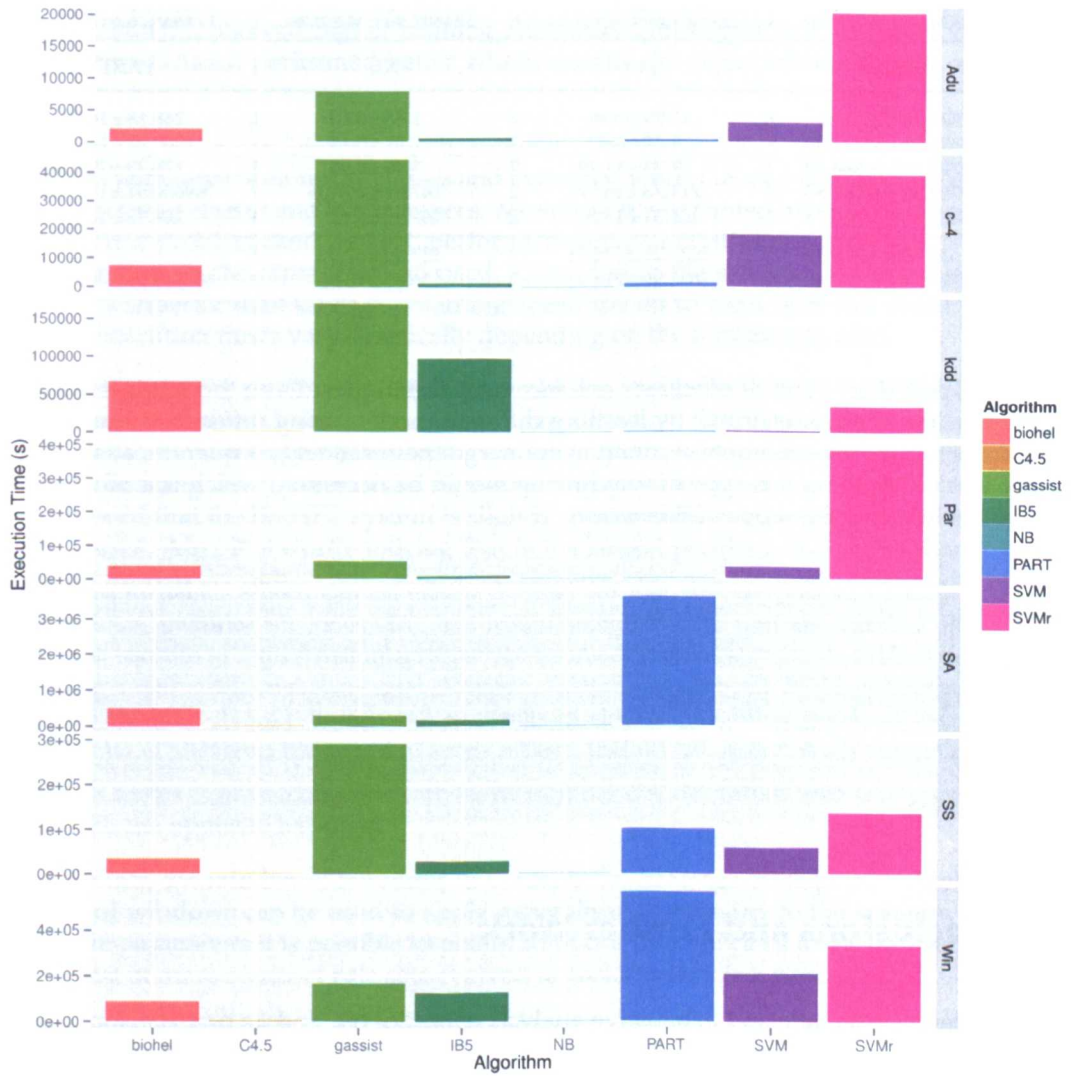


Figure 4.8: Execution time of the different ML algorithms over large problems

Table 4.22: Comparison of GAssist and BioHEL with other ML algorithms in terms of execution time (s). The configurations that obtained the best test accuracy in Table 4.21 are emphasised and the rankings within a particular problem are shown on the right

Problem	BioHEL		SVM		SVM _r		IB5	
Adu	1998.31±227.34	5	2987.77±536.19	6	20144.31±4244.21	8	450.20±156.87	4
c-4	7197.09±549.15	5	18218.58±3085.65	6	39210.92±2947.20	7	1096.81±180.30	4
Par	39422.64±9210.71	6	32354.57±5692.19	5	383081.72±58842.01	8	5562.40±65.50	4
SS	33430.59±14520.92	4	57132.69±12670.91	5	134243.30±17331.00	7	26407.76±3243.02	3
kdd	65806.99±15761.75	6	1346.72±299.21	4	32161.43±3106.12	5	96825.96±17485.70	7
SA	468821.16±64054.87	4	–		–		146577.52±13366.74	3
Win	88696.09±19758.60	3	207982.69±37931.41	6	324807.90±34278.76	7	123638.76±17691.04	4
Problem	GAssist		C4.5		NB		PART	
Adu	7875.33±1361.80	7	13.79±0.96	2	1.88±0.10	1	248.18±14.03	3
c-4	44531.39±3286.93	8	9.32±0.52	2	2.29±0.14	1	962.33±110.26	3
Par	52775.03±10888.52	7	21.23±11.29	2	3.40±0.90	1	174.23±290.39	3
SS	289782.58±21004.80	8	714.73±152.75	2	167.61±167.05	1	99650.51±19064.10	6
kdd	168481.84±20360.71	8	262.31±14.50	2	167.92±5.51	1	294.78±16.76	3
SA	271639.70±24879.49	5	6073.21±776.81	2	437.39±57.56	1	–	
Win	164214.74±12100.49	5	2873.90±214.96	2	156.85±18.96	1	562819.03±46249.84	3

does not support these kind of efficiency enhancement techniques. Even though it is possible to apply stratification (for example by learning different models using different subsets of the data and then applying ensemble techniques to merge these models), a quantification of the accuracy loss of applying this type of windowing would be necessary, which is a much more extensive study out of the scope of this work.

From this analysis we can conclude that the BioHEL system obtains results similar in accuracy to SVM and GAssist, but at a smaller computational cost. Moreover, the solutions provided by GAssist and BioHEL, being rule sets, are easily interpretable by human beings, in contrast with the models generated by SVM. Even though BioHEL generates larger rule sets, clustering techniques can be applied to obtain additional insights as shown in Bassel et al. [2011]. Considering the execution time and the accuracy, the BioHEL system seems to be the best candidate to solve large problems.

4.6 Conclusions and Further Work

In this chapter a systematic and exhaustive analysis over BioHEL and its predecessor GAssist was performed, in order to study the behaviour of the shared mechanisms applied to different paradigms. A standard configuration was determined for both systems. In the case of BioHEL, the best coverage breakpoint was 0.1 for small problems and 0.001 for big problems. In the case of GAssist, the optimal number of iterations was 1000 for small problems and 10000 for large problems. However, in the GAssist system no significant differences were found among configurations. Regarding the usage of the default class, the results differ between the two systems. While for the GAssist system the best results are obtained with the automatic policy, BioHEL obtains the best results with the disabled policy. Considering that this analysis was done over the small problems only, it was expected that the majority and minority policies did not stand out. When the problems are small, BioHEL finds it easier to learn rules from any class rather than limiting the possible classifier classes. However, the usage of the majority class is beneficial when solving large problems since this decreases the amount of examples the system needs to generate rules for.

After determining standard configurations for both systems, we compared them with the best configurations found per problem. We obtained significant differences in only two of the 28 small problems. Moreover, the percentage of accuracy loss in both systems was less than 3% in 25 of the 28 problems, and less than 6% in the rest of the problems. This indicates that the standard configuration found produce results similar to those obtained hand tuning the system. Moreover, when considering all the problems at the same time, there are no significant differences among the configurations, validating the previous hypothesis. However, for some problems, slightly better results can be achieved when the coverage breakpoint is set according to the problem.

When analysing the test accuracy against the training accuracy and number of rules in both systems, BioHEL showed higher training accuracies and bigger rule sets than GAssist. The cases where GAssist performed better were usually the cases where BioHEL overfitted the data.

Furthermore, when considering the characteristics of the problem, BioHEL performs better in problems with larger amount of classes and instances, while GAssist performs better with problems with fewer classes and less instances. Moreover, it was shown that GAssist performs better over discrete problems and BioHEL performs better over continuous problems. This is due to the differences in the representation used. Considering the execution times of both systems, BioHEL is superior than GAssist obtaining speedups up to 243X over this system. Nevertheless, the execution times vary drastically depending on the parameters used.

When analysing the results over the large problems, we showed again that BioHEL generates much more rules than GAssist. Moreover, the results show that the test accuracy and the number of rules are intimately related to the coverage breakpoint parameter in BioHEL, while in GAssist changing the number of GA iterations only produced major changes in two problems. This showed that the BioHEL system is slightly more sensitive to changes in the configuration than GAssist. Also, it is worth noticing that in the largest problems the most exhaustive experiments with GAssist (which are using 10 and 20 windows and 5000 and 10000 iterations) were not completed since they took an unpractical amount of time.⁷ On the other hand, BioHEL stands out in these scenarios, as it was designed for this purpose.

Increasing the number of windows in large problems produced a significant reduction in the execution time. Also the systems show a reduction in the test accuracy and the number of rules generated, especially in synthetic problems such as PMX. In GAssist, increasing the number of windows seems to have a bigger effect on the number of rules rather than in the accuracy, while in BioHEL the opposite happens.

Nevertheless, the number of iterations in GAssist, the coverage breakpoint in BioHEL and the number of windows can be used to apply generalisation pressure to the systems. Using more restrictive parameters it is possible to obtain solutions that are close to the best solution found, but consist of more compact rule sets evolved in considerably less time.

When comparing BioHEL and GAssist with other state-of-the-art ML algorithms taken from the WEKA package, our algorithms are ranked first and second, followed by SVM using gaussian (SVNr) and polynomial kernels (SVM). Significant differences were found between our systems and NaïveBayes, C4.5 and PART. Even though there are no differences in terms of accuracy between SVM, SVMr, GAssist and BioHEL, the latter provides the best accuracy/speed tradeoff, which suggest this system could be more adequate for this domain.

In summary, the tested EL systems are competent against other ML methods and, between them, GAssist is more suitable for small datasets, while BioHEL is more suitable for larger ones. Moreover, this chapter was not aimed only at identifying the best setting for each system, but also to study the behaviour of the shared components between the two methods. In this sense, we have shown that each system requires different policies for the usage of these components.

⁷These experiments were constraint by the limitations of our computational framework.

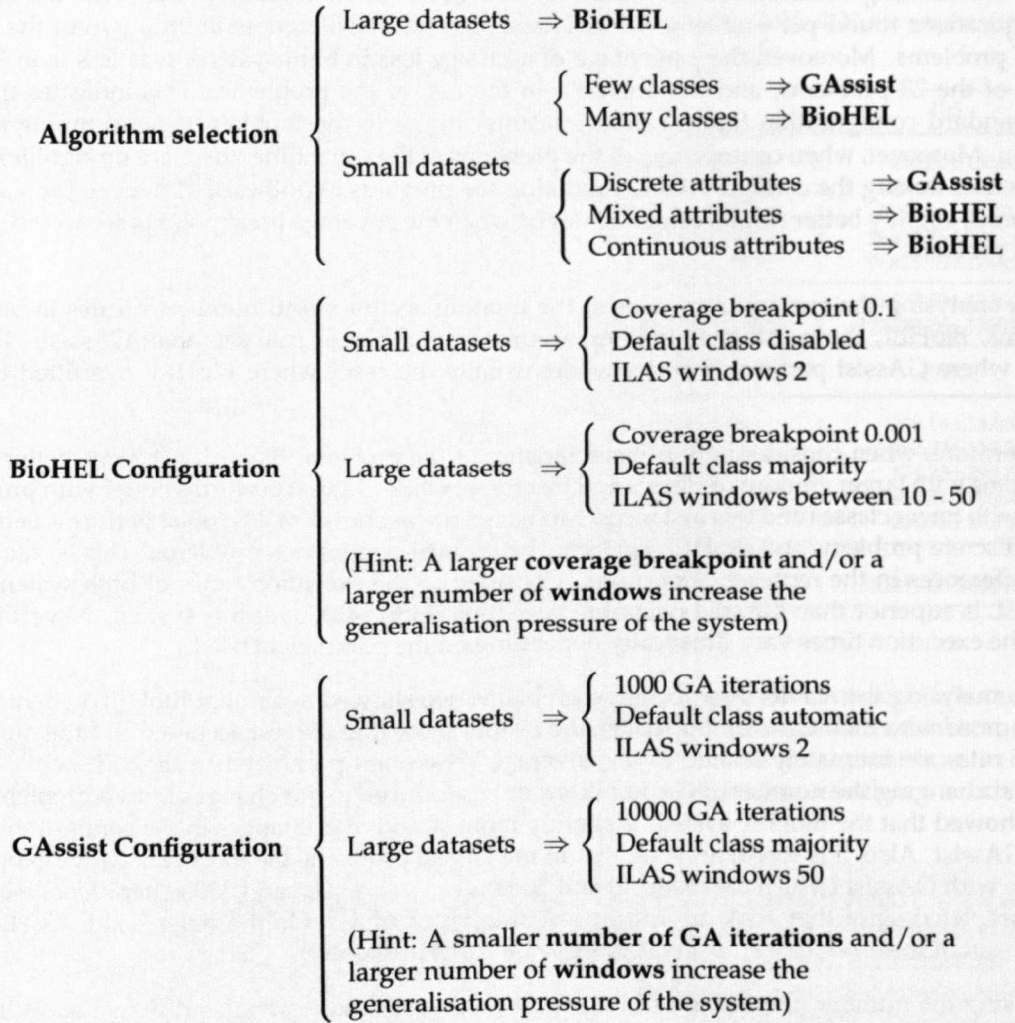


Figure 4.9: Algorithm selection scheme and recommended configurations for GAssist and BioHEL depending on the characteristics of the problem instances

In Figure 4.9 we present a decision tree for selecting the appropriate algorithm and the further system configuration depending on the characteristics of the problem.

Regarding the limitations of BioHEL encountered in this chapter we have the length of the final rule sets, which is considerably high compared to GAssist. In this sense we determined that it could be advantageous to apply techniques to reduce the cardinality of the rule sets. Moreover, the BioHEL system is very sensitive to the choice of parameters, and especially to the selection of coverage breakpoint, so it would be ideal to determine this parameter automatically. Finally, even though the BioHEL system is fast compared with other algorithms that obtain similar results, it could benefit from parallelisation techniques that have not been applied so far to speed up its learning process. These limitations are the motivation for the further improvements over the BioHEL system presented along this thesis.

Finally, based on the results obtained in this chapter, it would be interesting to analyse the ILAS Windowing mechanism thoroughly to determine when the usage of windowing becomes not beneficial in terms of accuracy. There are some models already for the GAssist system

that explain the behaviour of the windowing (See Section 2.4.1.3). These models assume the problems have rules that cover uniformly the whole search space. In the future, we would like to adapt these models to BioHEL, and cover problems where the rules are not uniformly distributed, do not have the same amount of expressed attributes and involve noise.

Parameter impact in BioHEL's fitness function

As shown in Chapter 4, BioHEL is highly parametrisable and better results can be achieved if the parameters are adjusted according to the problem at hand. However, it is still not clear how the adequate parameter setting is dependant on the characteristics of the problem. Therefore, this chapter presents an extensive parameter sensitivity analysis of BioHEL's fitness function. Specifically, we study two aspects of BioHEL's fitness function: its sensitivity to the coverage breakpoint parameter (that determines the degree of generality pressure applied by the fitness function) and the impact of the default rule policy (which changes the characteristics of the problem to learn). The results show that BioHEL is highly sensitive to the choice of coverage breakpoint and default class. While the right setting can facilitate the learning process, an incorrect setting can also push the system towards overgeneralisation. Moreover, it is shown how the coverage breakpoint parameter is closely related to characteristics of the problem itself such as the number of relevant attributes in the terms of the optimal solution. The conclusions of this chapter will lead to the proposal of a mechanism to adjust automatically the coverage breakpoint parameter later in the thesis.

5.1 Introduction

As it was explained in Chapter 2, BioHEL's fitness function is designed to generate accurate, general and simple rules (in terms of the compactness of the predicate). To this aim, this fitness function rewards the rules that cover (at least) a certain percentage of examples from the training set, by means of the user-defined parameter *coverage breakpoint*. However, this parameter is very problem-dependant and it requires an extensive preliminary experimentation to determine its adequate value, as it was shown in Chapter 4.

In this chapter we exhaustively analyse the impact of the coverage breakpoint parameter (with respect to the default class¹) in BioHEL's fitness function when learning challenging boolean synthetic problems, such as *k*-Disjunctive Normal Form problems. Since we can generate these problems with different number of relevant attributes (*k*) and different number of terms (*r*), it is possible to vary their difficulty on a regular step and determine how the optimal settings change with these small variations. The difficulty of these problems is based on the class imbalance and rule overlapping. If the *k* of the problem is small, the terms will cover a large percentage of the instance space and will tend to overlap. If *k* is large there is no overlapping but the positive examples would be very scarce, which also makes the problem difficult. Considering that the structure of the problem is known beforehand, we intend to asses the quality of the learning in terms of discovering this structure.

The analysis in this chapter is separated in two stages. First we test the performance and scala-

¹The usage of the default class can change completely the characteristics of the problem we are trying to solve as different sets of rules can be generated depending on the policy chosen.

bility of the system using different coverage breakpoints and default class policies. The aim is to identify the areas of the problem space that are more difficult to solve for BioHEL (in terms of number of relevant attributes k and number of problem terms r) for each coverage breakpoint value and default class policy. We present the results in terms of the iterations necessary to learn one of the k -DNF terms and the execution time. We also analyse the areas of the problem space where the system overgeneralises or it is not able to learn at all. The results also show the differences between using different default class policies. Even though the fixed class policy performs slightly better than the majority class policy, the latter shows how the learning would behave in situations where there is lack of crucial domain knowledge.

Moreover, these results show that depending on the characteristics of the problem the adequate coverage breakpoint value changes. Considering this, in the second part of the chapter an exhaustive experimentation is performed to determine the appropriate coverage breakpoint for each scenario, aligning the parameter values to the used representation. In this section we show a link between the characteristics of the problem (depending on the knowledge representation) and the adequate parameter value. This conclusion suggests the need to automatically determine these problem characteristics in order to set the parameters correctly.

The rest of the chapter is organised as follows. Section 5.2 presents the k -DNF boolean functions used in this study and the related work regarding the usage of these functions in the analysis of EL systems. Section 5.3 presents the experimental design and results of the preliminary parameter sensitivity analysis. Section 5.4 presents the second stage of experiments to determine the appropriate coverage breakpoint aligning the parameter values to the used representation. Finally, Sections 5.5 and 5.6 presents the conclusions and further work of this chapter, respectively.

5.2 k -DNF boolean functions

k -DNF (k -Disjunctive Normal Form) functions [Kearns, 1990] are a broad family of boolean functions which so far are a well known test suite in ML [Butz and Pelikan, 2006; Hernández-Aguirre et al., 2001] and in this thesis are used as a tool to characterise the structure of binary problems. Given a space of d attributes or variables the k -DNF functions are a boolean formula that presents the following form:

$$T_1 \vee T_2 \vee \dots \vee T_r \quad (5.2.1)$$

where r is the number of disjunctive terms and each term T_x represents the conjunction of k boolean variables out of the d possible options (x_1, x_2, \dots, x_d) , where some of the variables might have the *not* function (\neg) applied to them. Equation (5.2.2) represents an example of k -DNF function for a space of 10 representable attributes (d), 2 terms (r) and 4 represented attributes (k).

$$(x_1 \wedge x_5 \wedge \neg x_7 \wedge x_8) \vee (x_1 \wedge \neg x_3 \wedge \neg x_6 \wedge x_{10}) \quad (5.2.2)$$

Since EL systems learn a set of rules as the solution of a problem, this generalisation of binary problems into k -DNF formulas is really useful, as the systems are expected to learn one rule per term in the problem and the rules should represent at least all the relevant attributes in these terms. From this point onwards, every time we talk about “rules” we are referring to the solution of the problem and when we talk about “terms” we refer to the problem itself.

5.2.1 Dimensions of difficulty

The difficulty of a k -DNF problem can be estimated by calculating the class imbalance, which corresponds to the probability of finding a negative example in the dataset given its k and r values, as represented in Equation (5.2.3). This formula holds under the assumption that the

k attributes are randomly picked and hence there is no big overlap between any two terms. Figure 5.1 shows the corresponding probability distribution.

$$P(neg) = \left(1 - \frac{1}{2^k}\right)^r \quad (5.2.3)$$

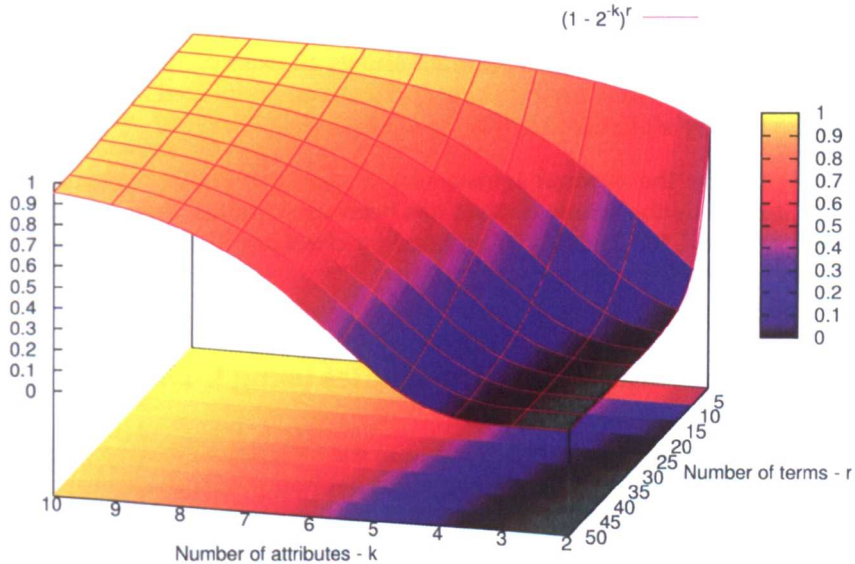


Figure 5.1: Probability of having a negative example in a k -DNF function according to the number of attributes k and the number of terms r

Since the class imbalance makes the problem more difficult, the red area in Figure 5.1 represents the problems that are easier to solve. Moreover, the class imbalance is not related to the size of the problem d . However, the size of the problem can increase the number of examples in the training set (as there are 2^d possible strings), and this variable can make the problem more difficult in terms of data volume.

As we can see from the previous formula the difficulty of the k -DNF problems depends on the number of relevant attributes k and the number of terms r . These two parameters can increase or decrease the number of positive examples in the training set and can potentially create scenarios with very high class imbalance. For example, when k is low each term covers a large proportion of the training set, and hence, just with a few terms (r), most of the examples will be positive. With moderate k values but high r the same occurs. In both situations we encounter a known source of difficulty in LCS: term overlap [Butz and Pelikan, 2006; Ioannides et al., 2011]. On the other hand, a high k and low r creates problems with very few positive examples, as each term is very specific and overlap is unlikely. These cases, also not balanced, essentially become scenarios of trying to find the needle in the haystack, because each term will cover very few examples.

We want to remark that this is an estimation of difficulty and represents the average case for randomly generated k -DNF problems. It can happen that for some problems depending on the configuration, there might be more or less overlapping which will produce slightly more or less negative examples. This is often the case in k -DNF problems with small d and a small k . The terms in these datasets cover a very large percentage of examples and hence it is not possible to generate smooth transitions of class balance.

5.2.2 k-DNF as a Machine Learning benchmark

The k-DNF problems used in this thesis were created using the generator at <http://icos.cs.nott.ac.uk/datasets/kdnf.html>. This generator works as follows. First, the user specifies the values d , k and r . Given this information r terms are generated, by randomly selecting k of the possible d attributes and checking that this combination has not been repeated already. Then, the *not* function is applied to each one of the k represented attributes in the term with 50% probability. Afterwards, all the possible 2^d binary strings (the instances of the dataset) are generated. They are assigned to class 1 if the string makes the formula true. Otherwise, they are assigned to class 0. The examples with class 1 are what we call “positive examples” and the rest are “negative examples”. Figure 5.2 shows an example of a generated k-DNF dataset in WEKA format. An example of a correct and an overgeneralized set of rules for this problem written in the ternary alphabet $\{1,0,\#\}$ is shown in Figure 5.3.

Given that we know the optimal solution for a specific k-DNF problem, in the experiments reported in this thesis the whole dataset has been used (rather than partitioning it into training and test sets) and the learning is assessed by checking that the correct k-DNF terms have actually been learnt.

5.2.3 Previous Machine Learning studies using k-DNF functions

The k-DNF functions have already been used in the ML context to determine the learning complexity [Kearns, 1990] and to derive lower and upper bounds for the sample complexity [Ehrenfeucht et al., 1988].² Moreover, they have also been used to derive learning models [Hirschberg et al., 1994] that predict the mean classification error of a given a distribution of the instance space.

Regarding GAs, [Hernández-Aguirre et al., 2001] used the Probably Approximately Correct (PAC) learning to estimate the initial population of the GA necessary to learn a k-DNF function with a given error. They show how the system evolves the solutions using the hamming distance of the individuals from the optimum. The results show that the population size needed to assure the existence of an individual close (similar) to the target chromosome is rather small.

Furthermore, this family of problems has already been used in the LCS context by Butz and Pelikan [2006]. In this work these functions were used to investigate the influence of replacing the crossover with other recombination algorithms based on EDAs. The goal is that the algorithm learns the intrinsic structure of the problem, in order to tackle more difficult problems like hierarchical ones. This research points out that one of the difficulties that LCS encounters when handling this type of problems is the term overlapping. This factor leads to the generation of rules that are partially but not maximally accurate, which makes the problem more difficult.

5.3 Parameter sensitivity analysis

This section is focused on performing an exhaustive experimentation using different values for the coverage breakpoint parameter and the default class policy. The experiments presented on this section are focused on the following two goals which correspond to independent result sections.

1. Determining the impact of the coverage breakpoint and default class policies in the learning process.

²The number of examples needed to learn a classification problem

```

% Rule 0 x0=1 && x3=0
% Rule 1 x1=1 && x4=0
% Rule 2 x1=0 && x2=0
@relation kDNF problem

@attribute x0 {0,1}
@attribute x1 {0,1}
@attribute x2 {0,1}
@attribute x3 {0,1}
@attribute x4 {0,1}
@attribute class {0,1}

@data
0,0,0,0,0,1
1,0,0,0,0,1
0,1,0,0,0,1
1,1,0,0,0,1
0,0,1,0,0,0
1,0,1,0,0,1
0,1,1,0,0,1
1,1,1,0,0,1
0,0,0,1,0,1
1,0,0,1,0,1
0,1,0,1,0,1
1,1,0,1,0,1
0,0,1,1,0,0
1,0,1,1,0,0
0,1,1,1,0,1
1,1,1,1,0,1
0,0,0,0,1,1
1,0,0,0,1,1
0,1,0,0,1,0
1,1,0,0,1,1
0,0,1,0,1,0
1,0,1,0,1,1
0,1,1,0,1,0
1,1,1,0,1,1
0,0,0,1,1,1
1,0,0,1,1,1
0,1,0,1,1,0
1,1,0,1,1,0
0,0,1,1,1,0
1,0,1,1,1,0
0,1,1,1,1,0
1,1,1,1,1,0

```

Figure 5.2: Example of a randomly generated k -DNF dataset with $d = 5$, $k = 2$ and $r = 3$.

Correct Rule Set	Overgeneralised Rule Set
1 # # 0 # 1 -> 100%	1 # # 0 # 1 -> 100%
# 1 # # 0 1 -> 100%	# # # # 0 1 -> 81%
# 0 0 # # 1 -> 100%	# 0 0 # # 1 -> 100%
Default Rule	Default Rule
# # # # # 0 -> 100%	# # # # # 0 -> 100%

Figure 5.3: Example of a correct and an overgeneralized rule set which solve problem in Figure 5.2. The number on the right represents the accuracy of each rule. 1 or 0 indicates that the attribute should have that specific value. # corresponds to an irrelevant attribute in the predicate (don't care).

2. Determining the areas of the problem space that are more difficult for BioHEL.

5.3.1 Experimental design

For these experiments we tested four different values for the coverage breakpoint parameter (0.1; 0.01; 0.001; 0.0001). We also performed two sets of experiments with different default class policies: a fixed class and the majority class. The first set of experiments used a fixed default class equal to 0 which corresponds to the negative examples of the problem. Since in the used k-DNF datasets all the terms map to class 1, this setting forces the system to learn exactly the terms of the function. For the second set of experiments we used the majority class as the default class. In this case the system might not learn the terms of the k-DNF function when the default class is 1 but instead it will learn the inverse problem, whose structure is unknown. These sets of experiments were designed to highlight the main differences in rule fitting when the default class policy changes. These experiments are also interesting because they will reflect how BioHEL will behave when handling a real problem, where we actually do not know which one is the most suitable default class.

The results in the first section will be presented in terms of how many iterations of the GA does it take to learn one of the terms of the k-DNF function. This will show if it gets harder for BioHEL to find good rules when the problem gets harder and if the coverage breakpoint can speed up or delay the learning process. This metric is only shown for the experiments with default class 0, as it cannot be computed in a straight forward manner when using the majority default class policy. When using the majority default class BioHEL is forced to learn the inverse problem when the amount of positive examples is higher than the negative ones. Even though the inverse problem can be calculated the characteristics of this problem in terms of k and r differ from the original problem, which is the particular difficulty over which we want to quantify BioHEL's effort. We also report the run-time of the overall learning process for both policies.

The second section is focused on determining the areas of the problem space that are more difficult for our system. For this purpose we generated learning maps. To determine the success of the system on each particular scenario we counted the number of final rule sets that fell in each one of the following categories:

Learning: the system learned the correct set of rules, in other words learned r rules which correspond to the r terms in the k-DNF problem and therefore, they have 100% accuracy.

Overgeneralisation: the system learned some of the k-DNF terms and generalised others. Even though, the system learned some good rules, the rule set as a whole does not have 100% accuracy.

No Learning: the system was not able to generate any rules besides the default rule.

Table 5.1: List of BioHEL's parameters different from the default configuration for the parameter sensitivity analysis. For a complete list of parameters refer to Section 3.3

Parameter	Value
GA Iterations	20
Initial MDL TL ratio	0.25
Number of strata in the ILAS windowing scheme	50

It is worth mentioning that the rule sets that present more rules but contain the correct ones, plus some additional terms also fall into the learning category. These are over specific rules which appear when the generalisation pressure is not high enough and they can be amended by making small local searches as it will be shown in Chapter 9. This classification of the rule sets can be considered a population-state metric (a metric that analyses the structure of the generated solutions [Kovacs, 2004] instead of the accuracy). However, the accuracy is still considered in this case to distinguish between overgeneral and correct rules.

The k-DNF problems used in this work have $d = 20$, $k = \{2..10\}$ and $r = \{5, 10, 15, 20, 25, 30, 35, 40, 45, 50\}$. This translates in 90 different scenarios of problem difficulty. For each one of these scenarios we generated 10 random problems and each one of them was run using 25 different seeds, so the results are the average of 250 runs.

Since the system has a stochastic behaviour some of the runs for the same problem scenario might have generated different rule sets and these rule sets might have fallen into different categories. For a specific scenario we considered that the system learns the problem when 90% of the 250 runs learned the correct set of rules. Moreover, we consider that the system does not learn if more than 5% of the runs were not able to generate any rules. The rest of the cases are considered overgeneralisation cases. In our experiments we have not observed any case where the conditions of learning and no learning were true at the same time.

In this chapter we used BioHEL with the base parameters presented in Chapter 3, with the exception of the parameters shown in Table 5.1. For a detailed explanation of these parameters please see Section 2.4.2. Moreover, the experiments were run in the High Performance Computing facility at the University of Nottingham each node with 2 AMD Opteron 2.2 GHz with 1GB per core.

5.3.2 GA iterations and execution time

In this section we measure how many GA iterations are required on average for BioHEL to learn each of the optimal rules of a k-DNF problem. We would like to remind the reader that BioHEL has two levels of iterations one embedded within the other. Rule learning iterations (as it learns rules sequentially) and GA iterations (as it uses a GA to learn each rule). For brevity, in this section whenever we use the word iterations we are always referring to GA iterations.

Figure 5.4 shows how many iterations BioHEL needs to learn each term of the k-DNF problem using default class 0 and different coverage breakpoints. For each one of the scenarios we only considered the runs where the system learned the optimal solution to the k-DNF problem. In this figure, the iterations needed to learn a term are proportional to k , the number of attributes expressed in each term, and are also proportional to r , the total number of terms to learn. Table 5.2 shows the best model that fits these points. A relationship between the number of iterations and the number of relevant attributes was expected since more specialised rules are more difficult to learn. However, this model also shows a relationship between the number of iterations and the number of terms. A possible explanation for this relationship is the overlapping between the terms. When the number of terms increases, the likelihood of overlapping increases

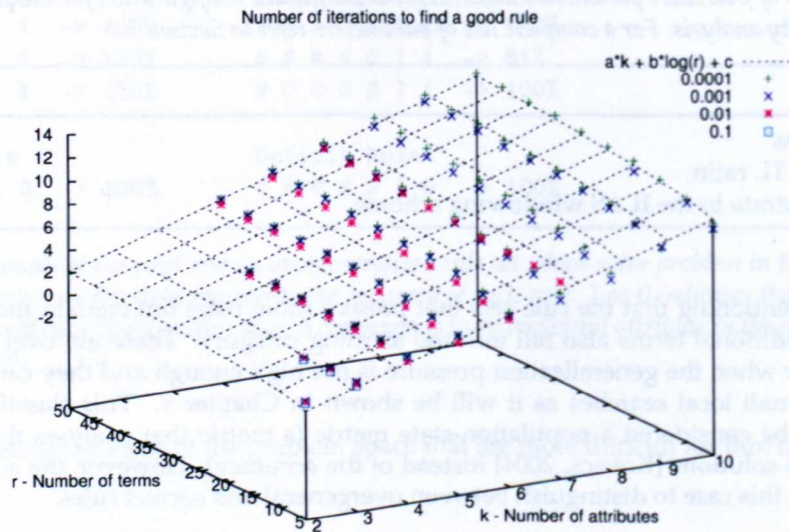


Figure 5.4: Number of iterations necessary to learn a good rule according to the number of attributes k and the number of terms r

as well and it takes more time to distinguish between the optimal and quasi-optimal (overgeneral) rules. Furthermore, when k is small it is possible that rules close to the optimal appear in the initial population from the start. This is the reason why for smaller values of k and r the number of iterations required to obtain a good rule with the required number of attributes is close to 0.

Table 5.2: Model of the number of iterations necessary to learn a good rule. P-value shows if the fitting of that particular variable is significant with respect to the data

Var	Value	Asymptotic Standard Error	% Error	P-value
a	1.39458	± 0.01586	(1.137%)	$< 2 \cdot 10^{-16}$
b	0.980511	± 0.04387	(4.475%)	$< 2 \cdot 10^{-16}$
c	-4.59467	± 0.1461	(3.181%)	$< 2 \cdot 10^{-16}$

Model: $iter = a \cdot k + b \cdot \log(r) + c$

Table 5.3 shows with greater detail the information presented in the previous graph. In this table it is noticeable that while the coverage breakpoint increases the algorithm is able to solve less problems. Also, the upper right corner is never solved because most of the instances in the training set are positive.³ In this area the system learns overgeneralised rules that cover bigger areas of the search space. Particularly, in the configurations with $r \in \{40, 45, 50\}$ and $k = 2$ all the examples are positive so the system learns only one rule that covers all the examples with 100% accuracy. Even though the accuracy is perfect, these cases were not considered when reporting the number of iterations, because the system was not learning the exact terms of the function. However, what the system is doing is not completely wrong as it is trying to solve the problem with a less complex rule sets.

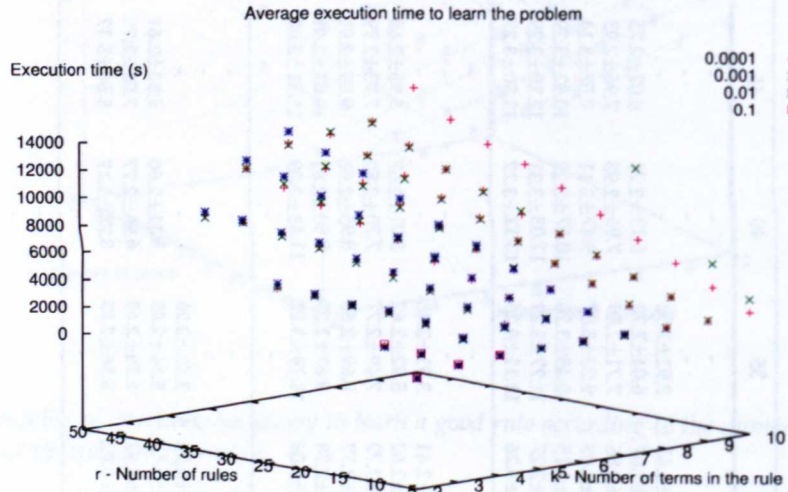
Furthermore, there are small changes in the number of iterations using different coverage breakpoints. By highlighting the best configuration (in terms of iterations) for each scenario in Table 5.3, we can see that the adequate coverage breakpoint varies among configurations. Moreover, there is a relationship between the shape of the emphasised areas and the proba-

³See class imbalance model for k-DNF problems in Section 5.2.

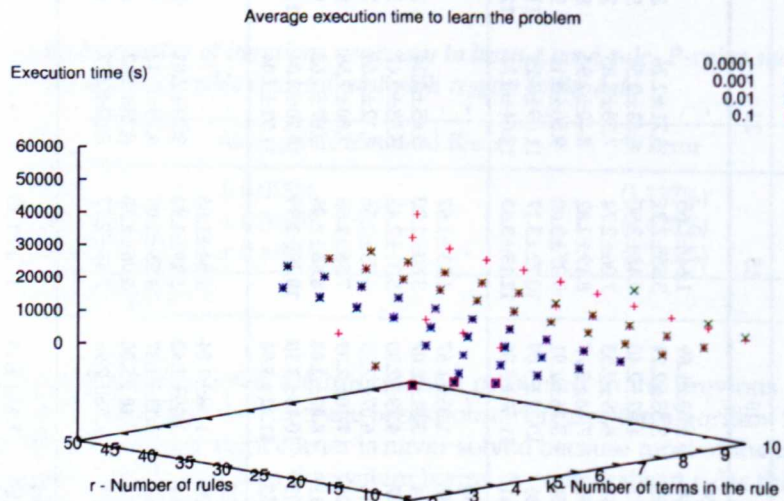
Table 5.3: Average iterations necessary to learn a term using different coverage breakpoints and default class 0. The highlighted data corresponds to the minimum number of iterations found for that particular k -DNF configuration

Coverage	k	Number of terms (r)									
		5	10	15	20	25	30	35	40	45	50
0.0001	2	0.62±1.24	-	-	-	-	-	-	-	-	-
	3	1.64±1.48	1.83±1.69	1.64±1.65	-	-	-	-	-	-	-
	4	3.25±2.00	3.55±2.11	3.65±2.21	-	-	-	-	-	-	-
	5	4.33±2.12	4.92±2.40	5.48±2.50	3.71±2.26	3.89±2.40	3.94±2.45	3.92±2.52	-	-	-
	6	5.60±2.31	6.39±2.55	7.06±2.74	5.63±2.58	5.93±2.65	5.96±2.67	6.04±2.70	6.13±2.74	6.07±2.75	6.26±2.87
	7	6.55±2.50	7.68±2.77	8.19±2.82	7.38±2.83	7.41±2.80	7.67±2.86	7.71±2.89	7.95±2.93	7.96±2.93	8.07±2.97
	8	7.58±2.57	8.76±2.91	9.37±3.00	8.40±2.90	8.84±3.02	9.05±3.03	9.22±3.07	9.47±3.11	9.57±3.14	9.63±3.18
	9	9.02±2.84	10.22±3.09	10.72±3.17	9.80±3.11	9.94±3.08	10.21±3.15	10.45±3.20	10.67±3.22	10.82±3.29	10.95±3.27
	10	10.65±3.41	11.57±3.36	12.15±3.33	11.09±3.23	11.45±3.24	11.64±3.27	11.77±3.29	12.03±3.31	12.10±3.29	12.24±3.31
					12.64±3.31	12.76±3.27	12.87±3.26	13.11±3.25	13.12±3.27	13.30±3.27	13.42±3.21
0.001	2	0.65±1.30	-	-	-	-	-	-	-	-	-
	3	1.64±1.47	1.83±1.67	1.53±1.67	-	-	-	-	-	-	-
	4	3.16±1.94	3.51±2.07	3.60±2.20	3.65±2.24	3.83±2.35	3.91±2.41	3.91±2.48	-	-	-
	5	4.31±1.97	4.79±2.30	5.31±2.42	5.53±2.47	5.87±2.58	5.91±2.62	5.92±2.62	-	-	-
	6	5.27±2.14	6.07±2.35	6.73±2.56	7.12±2.69	7.11±2.62	7.35±2.72	7.49±2.77	6.01±2.67	5.95±2.69	6.18±2.77
	7	5.96±2.05	7.10±2.39	7.58±2.55	7.80±2.58	8.30±2.71	8.45±2.77	8.69±2.82	7.70±2.80	7.73±2.79	7.81±2.85
	8	7.04±2.15	8.07±2.44	8.65±2.58	8.97±2.65	9.12±2.66	9.41±2.78	9.67±2.83	8.95±2.91	9.09±2.91	9.19±2.97
	9	9.18±3.03	10.10±3.10	10.39±2.99	10.70±3.04	11.00±3.06	11.11±3.06	11.19±3.02	9.90±2.87	10.02±2.91	10.21±2.98
	10	10.11±3.45	11.22±3.94	-	11.71±3.04	-	-	-	11.43±3.09	11.51±3.08	11.64±3.10
0.01	2	0.61±1.27	-	-	-	-	-	-	-	-	-
	3	1.48±1.28	1.68±1.54	1.54±1.66	-	-	-	-	-	-	-
	4	2.73±1.64	3.09±1.82	3.19±1.94	3.23±2.01	3.48±2.16	3.59±2.24	3.53±2.26	-	-	-
	5	3.64±1.53	4.11±1.87	4.55±2.01	4.75±2.12	5.20±2.39	5.29±2.45	5.34±2.45	-	-	-
	6	4.95±1.84	5.40±2.00	5.96±2.28	6.25±2.32	6.30±2.39	6.68±2.62	6.76±2.69	5.52±2.61	5.52±2.61	5.75±2.76
	7	7.53±2.99	7.88±2.96	7.88±2.77	8.02±2.77	8.35±2.95	8.46±3.02	8.56±3.03	6.96±2.77	7.04±2.78	7.30±3.04
	8-10	-	-	-	-	-	-	-	8.78±3.17	8.88±3.17	9.07±3.27
0.1	2	0.50±1.04	-	-	-	-	-	-	-	-	-
	3	1.29±1.06	1.45±1.35	1.39±1.38	-	-	-	-	-	-	-
	4	3.21±1.93	-	-	-	-	-	-	-	-	-
	5-10	-	-	-	-	-	-	-	-	-	-

bility of having negative examples shown in Figure 5.1. Furthermore, for each of the tested coverage breakpoint values, the region where a specific parameter value is the best is close to the areas where the system cannot learn the correct rules. This indicates that the coverage breakpoint value should be high enough to learn the problem quickly but low enough to avoid overgeneral rules.



(a) Default class 0



(b) Default class majority

Figure 5.5: Average execution time to learn a k -DNF function with different coverage breakpoints. Horizontally we present the coverage breakpoints tested and vertically the two default class policies.

The execution time of the learning process with both policies is shown in Figure 5.5. A significant model for the execution time could not be found but interesting patterns are present. As expected, using default class 0 the time increases with the number of terms the system needs

to learn but also increases with the number of attributes expressed in each rule. Considering that the number of iterations per GA run is fixed and remains the same for all the runs, we could attribute this behaviour to an increase in the execution time during the match process. Since BioHEL uses a representation where only the relevant attributes are represented in a list, when k increases, the match process becomes slower. What is more interesting is that there seems to be a relationship also between the time and the amount of negative examples in the problem. We can observe in Figure 5.5 a similar shape as the one obtained in Figure 5.1 around $k \in \{2, 3, 4, 5\}$. With the increase of negative examples the execution time increases, and particularly when using default class majority, there is a spiking behaviour on the area where there are more positive examples than negative ones. This is because, since the default class changes, the system is trying to learn a more difficult problem.

In general, the execution time does not seem to depend on the coverage breakpoint applied. However, there are some cases with $k = 10$ where there is a considerable difference in the run-time when using different coverage breakpoint values. These cases are very difficult to learn because, each time a rule is learned, very few examples are deleted from the training set. Using an inadequate (too low) coverage breakpoint might delay the process even more. Using default class majority we can see the execution time for some configurations (coverage breakpoint 0.0001 and low values of k) is higher than using a fixed default class. However, for the rest of the configurations the execution time is similar to using the default class 0. The reason is that for low values of k the system is learning the inverse problem which is more difficult than learning the exact k -DNF function.

5.3.3 Learning and overgeneralisation

Considering the number of cases where the system learned the complete set of rules, overgeneralised or did not learn anything at all, we generated learning maps as shown in Figure 5.6. The left part of the figure corresponds to the experiments with default class 0 while the right part corresponds to using the majority class as the default class. The figure shows 4 different states: Blue \square : good learning (the system learned the k -DNF terms in more than 90% of the runs), Red \circ : no learning (the system failed to generate rules in more than 5% of the runs), Green \triangle : overgeneralisation and (blank) incomplete scenarios (because the execution of each run took more than 10 hours). In the latter cases the runs were cancelled as the learning process was simply too slow and they are only present when using default class majority.

This figure shows that, for default class 0, the system tends to overgeneralise or simply not learn at all when using large coverage breakpoints over problems with large k . Additionally, it is noticeable that the two areas where the learning is more difficult are: a) the upper left corner (low values of k and high number of terms r) and b) the right side (larger values of k). The first region is very difficult because the number of positive examples is really high (see Figure 5.1). In this case the system tends to generate one or few rules that cover all the examples, degrading the accuracy slightly. The second region is difficult because there are few positive examples. In these cases, unless the appropriate coverage breakpoint is set up correctly (low enough), the system will generalise or not learn at all.

Regarding the usage of the majority class, using very small coverage breakpoint values produces difficulties in the area where the system is learning the inverse problem (low values of k). We can notice a large number of incomplete experiments. This occurs in the zone where the k -DNF problems are more balanced in terms of positive and negative examples but still the wrong default class is chosen. This is because the system is learning very specific rules that delete few examples from the population, but the actual amount of examples to learn is high. The area of incomplete experiments shrinks with the usage of larger coverage breakpoints. However, in these cases the system solves the problem overgeneralising.

The behaviour observed across increasing values of the coverage breakpoint parameter is sim-

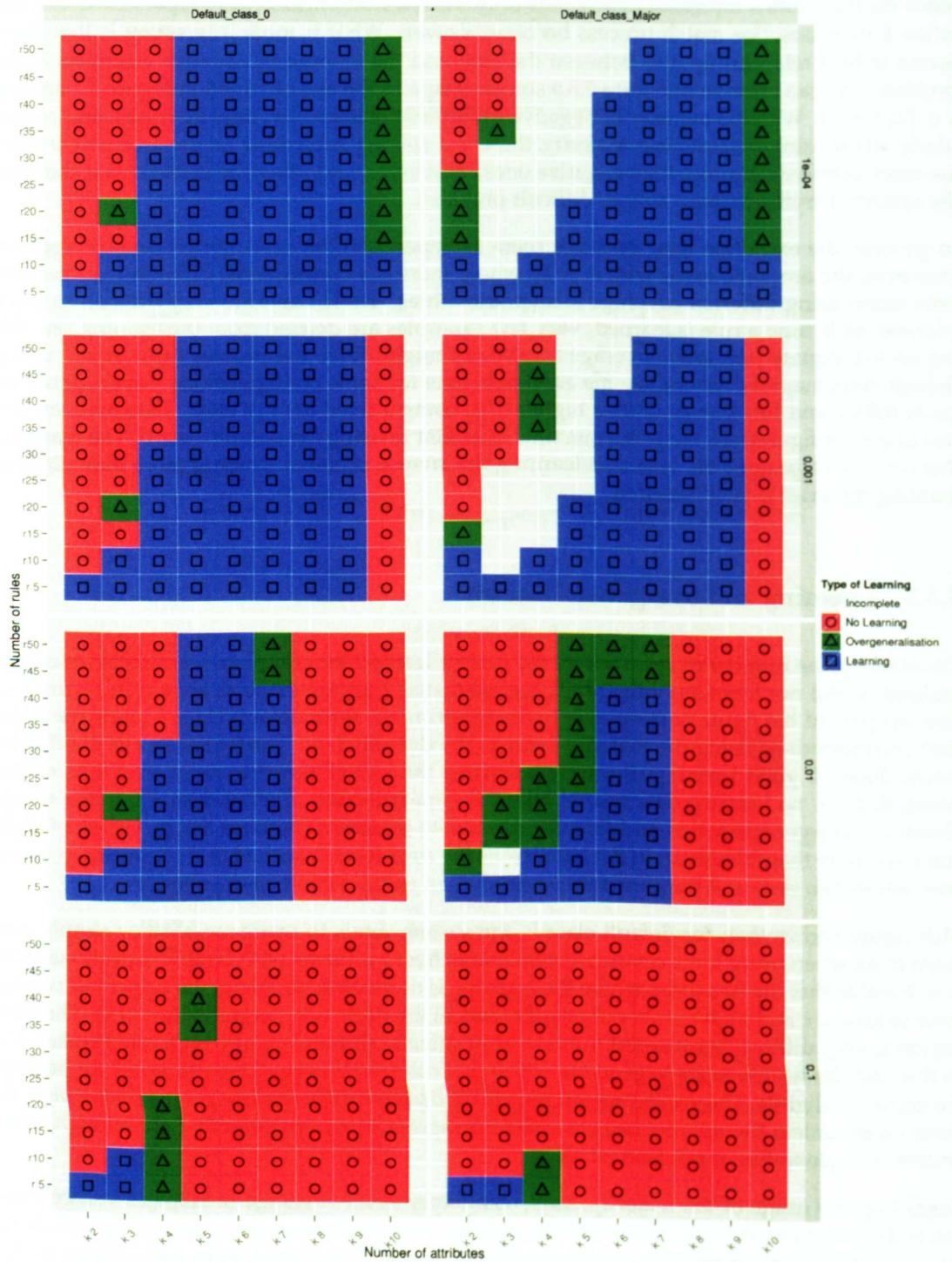


Figure 5.6: Learning and overgeneralisation maps using various *k*-DNF problems and coverage breakpoints (0.1, 0.01, 0.001 and 0.0001). Blue \square : good learning, Green Δ : overgeneralisation, Red \circ : no learning, blank: incomplete runs (runs that took more than 10 hours).



Figure 5.7: Learning and overgeneralisation maps aligned by coverage breakpoint values using various k -DNF problems, default class 0 and coverage breakpoints (0.1, 0.01, 0.001 and 0.0001). The groups represent different values of k (number of attributes). Blue □: good learning, Green △: overgeneralisation, Red ○: no learning

ilar. The areas of good learning shrink with the usage of larger coverage breakpoints. This indicates that a specific coverage breakpoint is necessary to solve certain scenarios.

Figure 5.7 shows the same results presented in the previous figure for default class 0, but tabulated per coverage breakpoint. In this figure it is clear that the right coverage breakpoint to solve the problem depends directly on k the number of attributes expressed. While k increases the minimum coverage breakpoint necessary to solve the problem decreases.

5.4 Towards a rationalisation of the coverage breakpoint settings

The previous section has shown, across a broad set of k-DNF variants, that each problem scenario requires a specific coverage breakpoint value. Moreover, something else was noticeable: this value is not easy to identify. As it is shown in Figure 5.6, the 0.01 configuration was able to solve just a fraction of the problems solved by the 0.001 configuration, and the 0.1 configuration only solved a very small fraction of the scenarios solved by the 0.01 configuration. This indicates that exploring the coverage breakpoint space using fractions of the training set size (10%, 1%, 0.1% of 0.01%) is not good enough to identify the optimal settings for every single k-DNF variant, as a single step change in the parameter can render the system unable to solve a large group of datasets.

Considering this, a better parameter exploration policy that avoids an exhaustive search is to align the explored coverage breakpoint values to the employed knowledge representation. For a binary representation, this means using coverage breakpoint values equivalent to the different number of expressed attributes the terms can have. If a term contains a single attribute (e.g. *If att_i is 1* \rightarrow *class = 1*) it will cover half of the search space, that is, a coverage of 0.5. If a term contains the conjunction of two attributes it will cover a quarter of the search space (a coverage of 0.25). Therefore, it would be much more efficient, for this representation, to explore coverage breakpoint values in the form of 2^{-x} being x the different numbers of attributes possible in the binary domain.

To this aim, the experiments were repeated on all the k-DNF datasets but in this case exploring coverage breakpoint values 2^{-x} where $x = \{2, 3, 4, 5, 6, 7, 8, 9, 10\}$. The learning status is reported in the same manner as in Section 5.3.3, since we are only interested in checking if BioHEL, using each of these coverage breakpoint values, can learn the evaluated datasets. As in the learning maps from the previous section, Blue \square represents the cases where more than 90% of the rules generated the correct set of rules. The Green \triangle represents the cases where the system overgeneralised the problem but learned some rules, and the rest of the grid (Red \circ) represents the cases where the system was unable to generate any rules in more than 5% of the runs.

Figure 5.8 shows the learning maps using aligned coverage breakpoints. As expected, using this parameter sweep policy it is possible to create much richer and incremental learning maps, as one step change in the explored coverage breakpoint values only changes a few cells in the learning map. Moreover, it is noticeable that, besides few exceptions, to solve a certain k-DNF problem the optimal coverage breakpoint setting is equal to 2^{-x} where x is equal or larger than the k of the dataset. The only drawback of using a coverage breakpoint smaller than needed is that more GA iterations are required to learn the optimal rule (as it was shown in the previous section).

There are few exceptions where this does not hold. These exceptions are the problems with high overlapping. In this situation, even when using the appropriate coverage breakpoint, BioHEL creates a single rule or a very small number of overgeneral rules, as these rules will still have very high accuracy. We can observe that for small values of k there is a limit in the number of terms the problem can have to be solved by BioHEL due to the high class imbalance. Table 5.4 shows the number of positive examples existing on average in the training set for the last solved scenario and the first unsolved scenario for $k=\{2,3,4\}$. In this table we can observe that if the number of positive examples is over 80% percent the system cannot solve the problem, but instead overgeneralises it.

We can also observe that in problems with $k > 7$ the overgeneral zone becomes smaller or non-existing. The learning map jumps almost directly from total learning to not learning. These problems are almost as finding the needle in the haystack. The wrong coverage breakpoint (larger than the size of the needle) renders the system unable to learn at all. An extreme case of

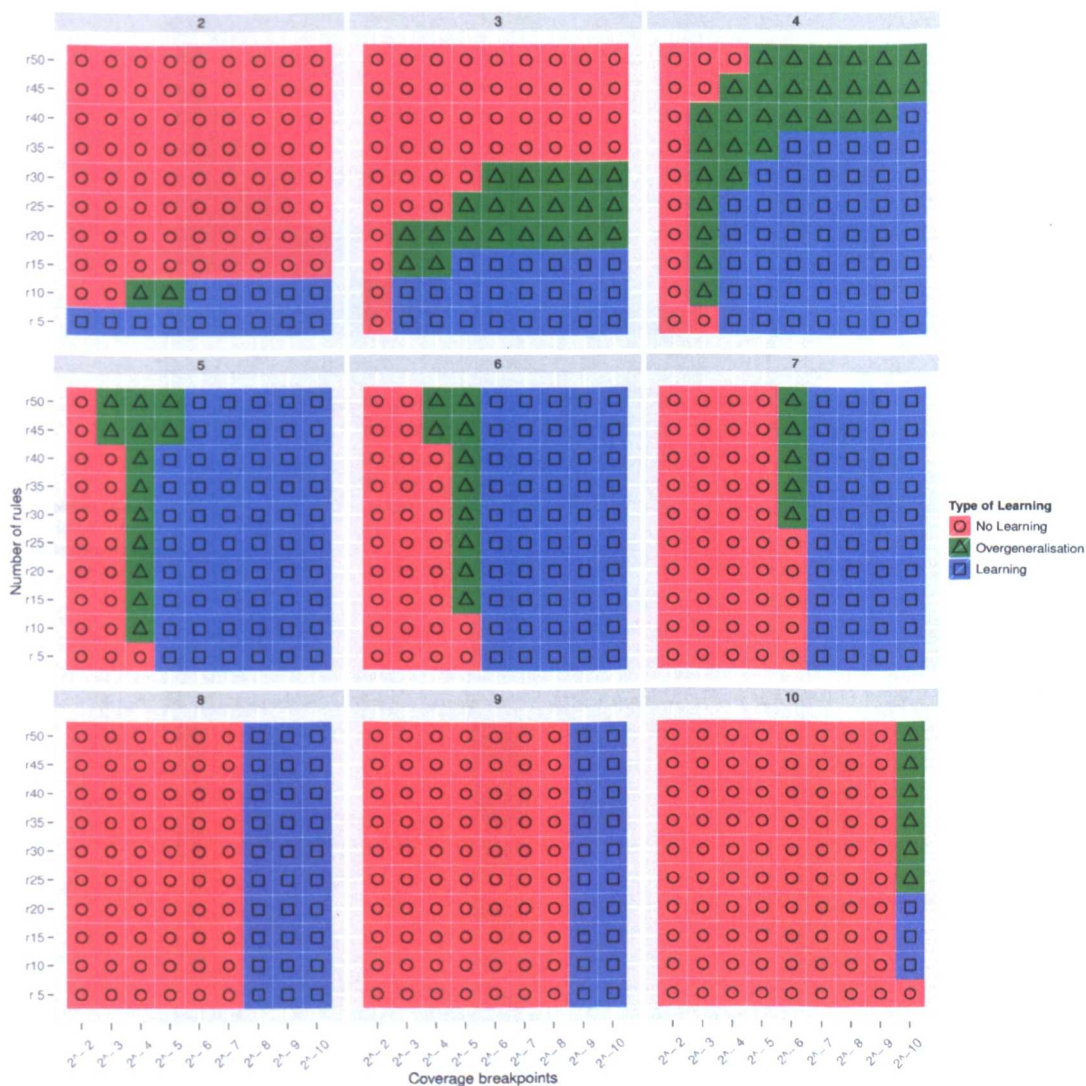


Figure 5.8: Learning and overgeneralisation map using various k -DNF problems and coverage breakpoints, where the coverage breakpoint is $1/2^k$ and the default class is 0. The groups represent different values of k (number of attributes). Blue \square : good learning, Green \triangle : overgeneralisation, Red \circ : no learning

Table 5.4: Percentage of positive examples for the last scenario solved and the first scenario not solved in BioHEL according to Figure 5.8.

Last Solved Scenario	Positive examples	First Unsolved Scenario	Positive examples
k=2 r=5	76%	k=2 r=10	94%
k=3 r=10	73%	k=3 r=15	86%
k=4 r=25	80%	k=4 r=30	85%

this phenomenon are the datasets with $k=10$, where not even the appropriate coverage breakpoint (2^{-10}) can solve the problem. In this case the ILAS windowing scheme is influencing the performance of the system. We know from our ILAS theoretical models (See Section 2.4.1.3) that in order for this mechanism to work well, each stratum should have representatives (instances) of all niches in the problem. That is, instances associated to all the terms in the problem's

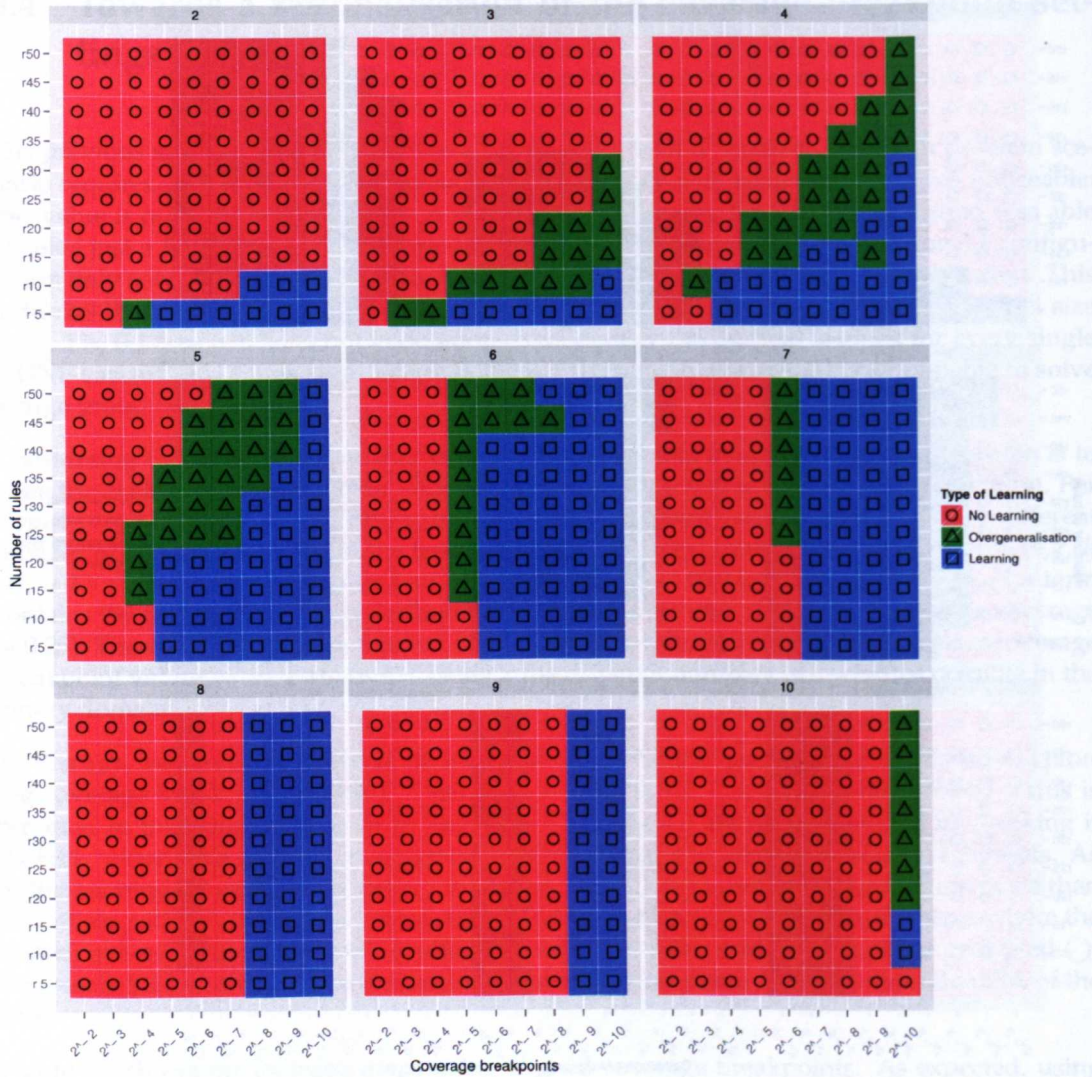
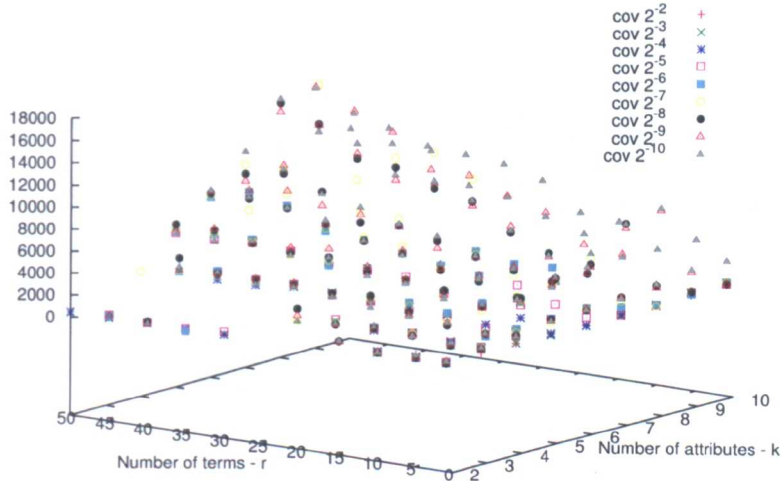


Figure 5.9: Learning and overgeneralisation map using various k -DNF problems and coverage breakpoints, where the coverage breakpoint is $1/2^k$ and the default class is the majority class. The groups represent different values of k (number of attributes). Blue \square : good learning, Green \triangle : overgeneralisation, Red \circ : no learning

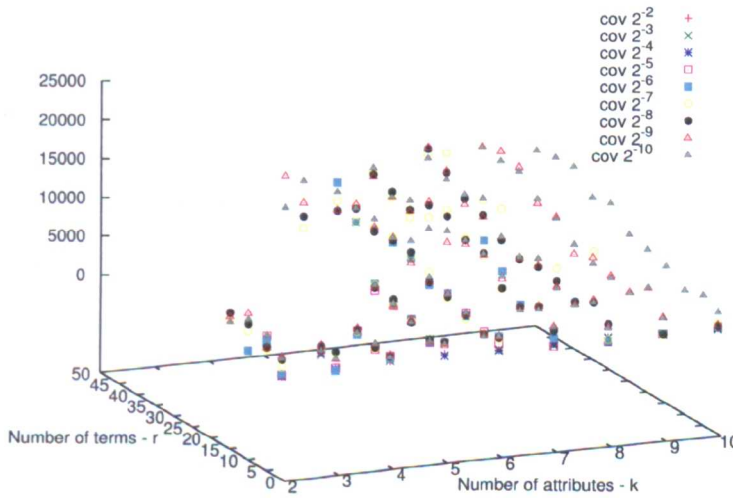
solution. When each term is so specific (as in the $k = 10$ case where each term is matching very few instances in the training set) it is very difficult that, during the ILAS strata generation, the random sampling process guarantees that each strata has enough representatives of all terms. To fix this situation the number of strata employed by ILAS should be reduced.

Also the same experiments with default class majority instead of only fixing it to class 0 were run and the results are shown in Figure 5.9. Similarly to the experiments in the previous section, the learning maps using a majority default class are slightly worse than the ones fixing the default class to 0. For larger values of k the results are similar to the ones using default class 0, since in these cases both versions are solving the exact k -DNF problem. For smaller values of k , the system tries to learn the inverse problem and much smaller coverage breakpoints are needed to at least learn an overgeneralised set of rules. The coverage breakpoint needed in these cases seems to be inversely proportional to the number of terms the problem has.

Figure 5.10 shows the execution time of the whole learning process using different coverage



(a) Default class 0



(b) Default class majority

Figure 5.10: Average execution time to learn a k -DNF function with different coverage breakpoints of the type 2^{-k} . Horizontally we present the coverage breakpoints tested and vertically the two default class policies.

breakpoint values and default class policies. Here we can observe that the execution time is not linked to the coverage breakpoint used and the spiking behaviour corresponds to using and incorrect parameter value in some cases which can make the learning slower. The same patterns observed in Figure 5.5 are presented in this figure for both default class policies. Considering that the adequate coverage breakpoint value for a k -DNF problem in 2^{-k} where k is the number of attributes expressed in the k -DNF terms, Figure 5.11 shows the average execution times for the optimal configuration in each scenario. In this figure it is clear that the execution time increases depending on k and r .

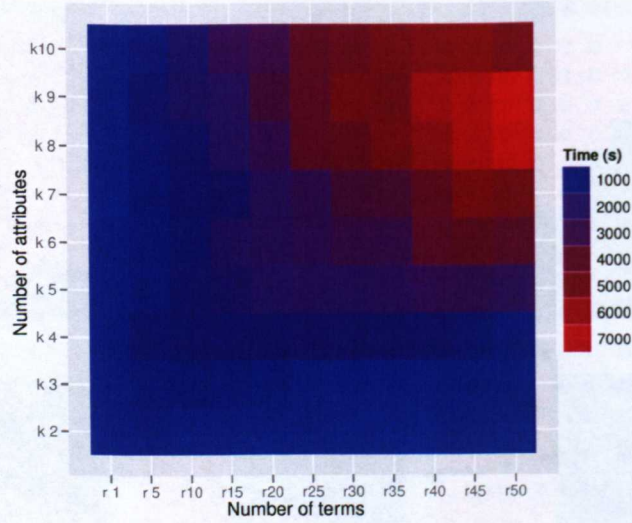


Figure 5.11: Average execution time to learn a k -DNF function using the optimal coverage breakpoint for each scenario

5.4.1 A general coverage breakpoint setting for binary domains

In this section we have observed that to solve a certain k -DNF problem we have to set up the coverage breakpoint to be equivalent to 2^{-k} where k corresponds to the number of relevant attributes in the problem. At this point it would be natural to think that what we have done is just to align the parameter specifically to the k -DNF problem at hand. This is not the case because our goal is to feed the system with structural information about the optimal solutions. For instance, the terms in the 6-bit multiplexer have a k of 3 (two address bits and one data bit), the terms in the 20-bit multiplexer have a k of 4, a k of 5 is present in the 37-bit multiplexer and so on. The 18-bit hybrid Parity-Multiplexer problem [Butz, 2006] has a k of 9, as this problem is composed by a 6-bit multiplexer where each of these “bits” is the result of a 3-bit parity problem.

Therefore, to solve problems with binary attributes it always makes sense to explore values of the coverage breakpoint parameter of the type 2^{-k} . Moreover, this does not only apply to synthetic problems but also to real-world ones. For instance, the rules generated for binary protein structure prediction datasets present consistent and very similar number of attributes [Bacardit et al., 2009b].

5.4.2 Coverage breakpoint setting policies for more complex domains

For datasets that use more complex representations like χ -ary discrete or continuous attributes it would not be enough to estimate the number of attributes represented in a term. For example, in a χ -ary discrete representation (more than two nominal values per attribute) and under the assumption that each term would only activate a single value per attribute, the coverage breakpoints to explore would have a shape of χ^{-x} , where χ is the cardinality of the attributes and x is the number of relevant attributes in the optimal solution. On the other hand, if a term activates all but one of the attribute values per attribute, we would need coverage breakpoint values of the shape $(\frac{\chi-1}{\chi})^x$. In general, if the average number of values appearing per attribute in a term is e , the explored coverage breakpoint values would be $(e/\chi)^x$. This value may not be that easy to estimate in a global way. For instance, in some χ -ary protein structure prediction datasets

with a cardinality of 20 values per attribute [Bacardit et al., 2006], some of the attributes would have an e value close to 20 while other attributes would require an e value around 7.

In the case of a hyper-rectangle-based continuous representation, if each (specified) side of the hyper-rectangle occupies a fraction r of the attribute's domain, the coverage breakpoint values to explore would be in the form of r^x where x is the number of relevant attributes per term.

Overall, the problem of setting up the coverage breakpoint parameter is not solved, but it is translated to determining the structural characteristics of the problem based on the representation used. Therefore, a parameter sweep process that considers these structural characteristics would be much more effective.

5.5 Conclusions

In this chapter we have studied more in-depth the behaviour of the BioHEL evolutionary learning system using very diverse k-DNF boolean functions, which present different challenges to the BioHEL system, such as high class imbalance and term overlapping.

The experiments showed that the execution time and the number of GA iterations necessary to learn one of the k-DNF terms depend on the number of attributes expressed k and the number of terms r . The execution time of the algorithm does not depend on the coverage breakpoint used. However, the usage of the majority class can increase the execution times when the wrong default class is selected. It was shown that the default class policies are very helpful when the problem that we want to solve is unknown. Nevertheless, using the majority or minority class might not always be the best policy. When the problem is balanced using the majority or the minority class only discards half of the examples, which means that, if we are using a small coverage breakpoint, the learning will be very slow. The system starts generating plenty of rules that cover few examples, instead of generating a maximally accurate and general solution to the problem.

In this chapter we have gained some insights about the parametrisation of BioHEL's fitness function: a) each problem configuration needs a particular coverage breakpoint in order to solve the problem and b) the performance of the algorithm depends on the adequate selection of the coverage breakpoint. These problems get aggravated when trying to solve a real problem, since it could be very difficult to determine which is the adequate value for this parameter. The coverage breakpoint should apply enough pressure to learn rules that cover a considerable amount of examples but without forcing the system to overgeneralise. According to our experiments the adequate coverage breakpoint for boolean problems with k attributes expressed is 2^{-k} . For problems with few positive examples even smaller coverage breakpoints are required to force the system to learn the correct set of rules. Moreover, when there is high overlapping (more than 80% of positive examples), even having the right coverage breakpoint is not enough to solve the problem. At this point the generalisation pressure is too strong and the system tends to overgeneralise.

To avoid an exhaustive exploration of the coverage breakpoint space it is necessary to align the explored values to the characteristics of the knowledge representation and, if this knowledge is available, to the characteristics of the problem itself (number of relevant attributes, the sizes of the intervals for continuous attributes or the number of values per attribute on the χ -ary discrete representations). Still, in many cases this information is not available and a parameter sweep process would be required.

5.6 Future research directions

Towards the aim of determining the characteristics of the problem, Chapter 6 will describe theoretical models that estimate the probability of generating a good initial population of our ALKR+GABIL knowledge representation based on the problem characteristics (k and r). Afterwards, using this model and the negative instances model presented in this chapter it is possible to estimate the problem's k and r , and hence derive the appropriate coverage breakpoint value, as it will be described in Chapter 7.

Regarding the extreme cases, where the class imbalance is too high it would be interesting to revisit the work of Orriols et al. [Orriols-Puig and Bernadó-Mansilla, 2008a; Orriols-Puig et al., 2009]. This work develops class imbalance models for XCS which perhaps could be helpful to improve the BioHEL system. Moreover, it would be interesting to determine the impact of the ILAS windowing scheme sampling over the problems that have very few positive examples. Perhaps it could be possible to generate models that indicate the maximum number of windows that ensures the learning, considering the class imbalance.

In general other techniques such as granular clustering [Pedrycz and Bargiela, 2002, 2012], min-max clustering [Bargiela and Pedrycz, 2013; Simpson, 1993], and inclusion/exclusion clustering [Bargiela et al., 2004] seem to hold the key to handle imbalanced data. In these techniques the data is iteratively clustered together in *hyperboxes* (which are very similar to the hyperplane representation of BioHEL). Depending on the technique used, the hyperboxes are generated in different ways, but some concepts remain present: a) the membership or compatibility notion that allows the hyperboxes to merge and expand while still trying to be as compact as possible, and b) the concept of overlapping between classes. This allows to generate a representation that simply adapts itself to the underlined model of the data (with hyperboxes that can vary their size in different parts of the search space). Since these approaches are also supervised learning approaches and they use a representation somehow similar to BioHEL's hyperplanes, it would be very interesting to analyse these techniques in greater depth and try to apply these concepts to BioHEL to improve its performance on imbalanced data. For example our current rules can simulate hyperboxes and this hyperboxes can undergo a reorganisation phase (they can be merge or split if they are too big, according to the previously mentioned clustering concepts).

It would also be interesting to compare the k-DNF results obtained with BioHEL with other ML systems, although the choice of a common performance metric across multiple learning systems (e.g. XCS [Wilson, 1995], C4.5 [Witten and Frank, 2005], etc.) is far from simple. As part of these experiments, we would like to compare the learning map of other systems to see if they show the same patterns as BioHEL. Moreover, as we have shown that these datasets present several interesting challenges, we believe that it would be interesting to use them as standard benchmarks for EL.

Modelling the Initialisation Stage of BioHEL

Models are commonly used to gain understanding of how EL systems work. To understand better how the structure of the problem influences the behaviour of BioHEL, in this chapter we propose models for the probability of having a good initial population in BioHEL for problems with discrete attributes; this means using the Attribute List Knowledge Representation (ALKR) and the GABIL encoding. We base our work in the schema and covering bound models previously proposed for XCS. These models are extended to: (a) deal with the combination of ALKR+GABIL representation, (b) explicitly handle datasets with niche overlap and (c) model the impact of using the covering and default rule mechanisms in the representation. The models in this chapter are based on the problem characteristics and allow us to evaluate the challenges presented by problems with high cardinality (in terms of number of attributes and values of the attributes) as well as the benefits contributed by each of the components of BioHEL's representation and initialisation operators. Finally, since the models are based in structural characteristics of the problems, they constitute the basis to determine this structure and adapt the coverage breakpoint automatically.

6.1 Introduction

As it was shown in Chapter 5, BioHEL is highly parametrisable and the adequate parameter values depend on the structure of the problem at hand. However, it is still not clear how exactly this inner structure of the problem makes the learning more difficult for BioHEL. Moreover, the structure of the problem is unknown in real situations and we need to guess this structure to be able to parameterise the algorithm correctly. In order to gain precise knowledge about how the structure of the problem affects the learning process in BioHEL we performed an analysis of the initialisation stage.

Facetwise analyses (which include analysis of the initialisation stage) have been performed in the past over EL systems [Butz et al., 2004; Butz, 2006; Goldberg, 2002; Orriols-Puig, 2008; Orriols-Puig et al., 2010; Stalpf et al., 2012] to understand their domains of competence and the requirements that should be fulfilled for their correct functioning. Using this methodology we aim to determine the structure of the problem by observing the system's behaviour during the initialisation.

In this chapter we develop models that extend the covering and schema bounds proposed for the XCS system [Butz et al., 2004; Butz, 2006], that is, the probabilities that an initial population covers the whole search space and that it contains representatives from all niches in a problem, respectively. These models were originally created for the ternary representation $\{0,1,\#\}$, thus we extended them to work with BioHEL's binary representation: ALKR which, for discrete variables, uses the GABIL encoding (See Section 2.4.2.1). Moreover, the models explicitly take

into account other characteristics of BioHEL such as its covering operator and the use of an explicit default rule. Afterwards, specific models are developed for problems where there is overlap between some of its niches (problem terms). Finally, these models are generalised to deal with χ -ary attributes, where the cardinality of each attribute is greater than two.

These models are empirically evaluated, in a first stage, using binary problems with and without niche overlap. The experiments show the models are accurate in these two different scenarios. Moreover, the models also show how the covering and default rule mechanisms of BioHEL increase the probability of having a good initial population. In a second stage we validate those models for χ -ary attributes. The models show how the problem gets harder with the increase of the cardinality of the problem attributes and the number of classes. Nevertheless, they also show how the GABIL encoding can be more robust when the number of values per attribute increase, since this decreases the probability of generating unmatchable rules, a known weakness of this representation [Llorà et al., 2007].

Finally, using the probabilistic models derived we calculate the corresponding schema and covering bounds and show how the minimal theoretical population size for BioHEL changes depending on the characteristics of the problem.

These models are useful, not only because they show insights about the strengths and weaknesses of BioHEL, but also because they are the start point to design principled methodologies to automatically adjust some of the parameters of the system. This adjustment is particularly desirable when handling large scale datasets, since it helps avoiding the high computational costs involved in preliminary experimentation. A final discussion presents briefly how this models can help discover the structure of the problem if the behaviour of the system on its early stage is analysed. These models lead to the implementation of a heuristic to set the coverage breakpoint parameter automatically which is one of the main contributions of this thesis, shown in Chapter 7.

This chapter is structured as follows. Section 6.2 presents the previous work around modelling GAs and EL systems, with special emphasis in the schema and covering bound models already derived for XCS. Section 6.3 presents the derivation and validation of the probabilistic models for the binary domain in which the schema and covering bounds are based. Section 6.4 presents the extension of the models for the χ -ary representation. Section 6.5 presents the derivation of the schema and covering bounds from the probabilistic models derived. Section 6.6 presents a brief discussion about how to use the models to derive the problem structure. Section 6.7 presents the final remarks and Section 6.8 presents the further research directions of this chapter.

6.2 Evolutionary Learning modelling methodologies

Since GAs [Holland, 1975] were introduced by Holland in the 60's theory was developed to explain why and how GAs work. Holland first presented the definition of a *schema* which is a subset of strings with similarities over a number of inputs or positions. In a binary domain, a schema can be written as a sequence of ternary symbols that specifies some of the bits in a string of size d (i.e. $1*0*11$) [Holland, 1975],¹ where d corresponds to the total number of attributes or inputs of the problem, each bit corresponds to an attribute and the "*" corresponds to the attributes that are not relevant or can take any value. For example the schemata $1*0*11$ groups the following four strings {100011, 110011, 100111, 110111}. Moreover, the schemata can be used to represent partial solutions to the problem and the subset of schemata that correspond to the inner structure or terms of the problem are known as the *building blocks*.

Based on this theory Goldberg [2002] proposed an approach to formalise the design and ap-

¹We use the ternary representation to explain the concept of schema as it is more compact. However, the representation used in BioHEL for nominal attributes is the GABIL representation.

plication of GAs by employing a facetwise methodology in which the different aspects of the system are analysed individually, assuming that the rest of the components are working correctly. According to Goldberg the problem of designing a competent GA can be decomposed into the following seven subproblems:

1. Know what GA process (building blocks)
2. Know thy building block challengers (building-block-wise difficult problems)
3. Ensure an adequate supply of raw building blocks
4. Ensure increased market share for superior building blocks
5. Know building block takeover and convergence times
6. Make decisions well among competing building blocks
7. Mix building blocks well

This methodology was adapted by Butz et al. [2004]; Butz [2006] to the specific context of Michigan LCS. In recent years, this methodology has been applied to specific, challenging, learning scenarios such as problems with class imbalance [Orriols-Puig, 2008] or continuous domains [Orriols-Puig et al., 2010; Stalph et al., 2012]. The work developed in this chapter is based on the principles found in [Butz, 2006; Goldberg, 2002].

Furthermore, other formal analyses of LCS have been proposed from a more probabilistically model-based perspective [Drugowitsch, 2008; Edakunni et al., 2009]. In this work the authors model LCS as a Mixture of Experts (MoE), showing how this metaphor can generate a very similar prediction models and explain how LCS works from a ML point of view.

Also, other analysis over the initialisation stage of the GABIL representation have been performed in the past [Bacardit, 2005] in the context of the GAssist. In this work the probability of covering the search space with the GABIL representation was modelled in order to propose smart initialisation strategies for GAssist. Moreover, the GABIL representation has also been analysed in terms of scalability in [Llorà et al., 2007]. In this work the authors point out weaknesses of the GABIL representation, such as generating rules that are incapable of matching any example. Here the authors show how the number of unmatchable rules increase exponentially with the problem size, presenting scalability problems for the systems that use this representation.

6.2.1 Facetwise analysis of the initialisation stage

To solve the third problem presented by Goldberg (ensure an adequate supply of raw building blocks), there are two characteristics a good initial population needs to comply with, according to Butz et al. [2004]; Butz [2006]. First, the building blocks of the problem should be present in the initial population, meaning that there should be representatives of each niche (terms of the problem). That is rules that belong to a niche and do not misclassify. Moreover, the whole search space should be covered. These two requirements are referred as the schema and covering bounds respectively. In the following sections we present the corresponding probabilistic models for the XCS classifier system using ternary representation.

6.2.1.1 Schema Bound

A *representative* is a classifier that specifies at least all the relevant attributes in a problem schema. The representative, however, may be more specific than the schema, as some of the irrelevant attributes can be specified as well. For example if a problem schema is 1^*0^{**} possible

representatives would be 1#0## or 110#1. If we consider the schemata as the niches or terms of the problem, the representatives are the rules that do not make mistakes and therefore, have information about the building blocks of the problem.

The schema bound constraints the parameters of the system so the probability of having representatives of each schema in the initial population is high enough [Butz et al., 2004; Butz, 2006]. This is important since having good building blocks in the initial GA population is essential for the learning process [Goldberg, 2002], as it avoids falling into local optima. It is worth mentioning that it is also possible to generate representatives along the learning process, through crossover and mutation. However, the probabilities of this happening are enclosed in the reproductive opportunity (probability of generating representatives along the learning process) and sustenance (probability of survival of the good individuals) bounds, which are not related with the initialisation stage.

The schema bound is based on modelling the probability of generating a rule that is a representative of a schema. For the ternary representation this is:

$$P(rep) = \frac{1}{n} \left(\frac{\sigma[P]}{2} \right)^k \quad (6.2.1)$$

where k is the number of relevant attributes or bits represented in the schemas, n is the number of classes and $\sigma[P]$ is the specificity probability of the population. In initial populations this value is equal to $1 - P_{\#}$, the probability of specifying a value in a randomly generated rule, while later in the run it converges to 2μ , where μ is the mutation probability. In [Butz, 2006] the k is also regarded as the *minimal order of difficulty* of the problem.

Moreover, the probability that a certain niche is represented is equal to Equation (6.2.2) where N is the population size.

$$P(rep \text{ exists}) = 1 - \left(1 - \frac{1}{n} \left(\frac{\sigma[P]}{2} \right)^k \right)^N \quad (6.2.2)$$

From this formula it is possible to derive lower bounds for the population size N , and the specificity of the population $\sigma[P]$ that ensure that the probability of having the representative in the population is high enough. After deriving the schema bound for the population size it was shown that the smaller the value of $\sigma[P]$ the smaller the population size needed. Moreover, the population size grows exponentially according to k .

6.2.1.2 Covering bound

The covering bound assures that every possible instance of the problem is covered at least by one classifier in the population. In this way, we will map the whole problem domain into an initial population. The probability of this happening depends on how large and how general is the initial population.

To calculate the covering bound we need to calculate the probability of matching an instance with a randomly generated classifier. According to Butz [2006], in a uniformly sampled problem space with d representable attributes, the probability of matching an instance with a randomly generated classifier is:

$$P(match) = \left(2 - \frac{\sigma[P]}{2} \right)^d \quad (6.2.3)$$

Moreover, the probability that an instance is matched by a randomly initialised population of size N is:

$$P(\text{cover}) = 1 - \left(1 - \left(2 - \frac{\sigma[P]}{2}\right)^d\right)^N \quad (6.2.4)$$

Since the covering is only applied when the rules are initialised, the specificity of the population $\sigma[P]$ can be considered equal to $1 - P_{\#}$. Based on the previous formula, and using the same methodologies as with the schema bound, it is possible to derive boundaries for $\sigma[P]$. In Butz [2006] it was shown that the population size increases linearly with the average specificity $\sigma[P]$ and the size of the problem d .

6.3 Probabilistic models for ALKR+GABIL

In this section we will focus on calculating the probabilities on which the schema and covering bounds are based for the binary case using the ALKR+GABIL representation. We will also derive formulas for cases with overlapping for the schema bound. Afterwards, in Section 6.4 we generalise the models for χ -ary attributes. These probabilities can be used to specify restrictions or bounds over system parameters, as it will be shown explicitly in Section 6.5.

6.3.1 Schema Bound: Probability of generating a representative

The schema bound is based on modelling the probability of generating a rule that is a representative of a schema. The probability of a representative was already calculated for the ternary alphabet as shown in Section 6.2.1 based on the global specificity of the population and without considering overlapping. However, the probability for BioHEL is different since this system uses the ALKR+GABIL encoding. The probability for BioHEL depends also on the methodologies this system uses to create new classifiers (covering and default class) and two user-defined parameters p and $ExpAtts$ (See Section 2.4.2).

Considering the covering and the default rule mechanisms in BioHEL, there are four ways in which a rule can be created:

- (a) not using covering or default rule (base case)
- (b) using default rule only
- (c) using coverage only
- (d) using coverage and default rule

For each one of these cases we derived the probability of having a representative in problems that do not have overlapping. Afterwards, based on this probability we handle the niche overlapping case.

To calculate the probability of the representative we need to consider two types of attributes:

Fully mapped attributes. Attributes for which all its possible values are specified within the schemata.

Partially mapped attributes. Attributes for which only one of its possible values is specified in the schemata, either 1 or 0.

We will refer to the number of fully mapped attributes as k_f and the number of partially mapped attributes as k_p . If the schemata have k attributes, then $k_f + k_p = k$.

Considering that BioHEL uses the GABIL representation embedded within the ALKR, an attribute is relevant if it was selected to be in the attribute list. This occurs with probability l_d as shown in Equation 7.3.3. Moreover, an attribute can also be relevant if the attribute is fully mapped and the strings generated are 01 or 10 (equivalent to 1 and 0 in ternary encoding), or if the attribute is partially mapped and the “right” string is generated.

$$l_d = \begin{cases} 1 & d \leq \text{ExpAtts} \\ \frac{\text{ExpAtts}}{d} & d > \text{ExpAtts} \end{cases} \quad (6.3.1)$$

In the following sections we will show the final probability formulas for each one of the cases mentioned before.

6.3.1.1 Base case (No covering and no default rule)

Considering that a representative should have at least the number of bits in the schema represented k , the size of the problem d , and the number of possible actions n , the probability of a rule becoming a representative is:

$$P(\text{rep}) = \frac{\left(r_d^p\right)^{k_p} \left(r_d^f\right)^{k_f} (1 - l_d P(00))^{d-k}}{n} \quad (6.3.2)$$

where r_d^p is the probability of relevance of a partially mapped attribute and r_d^f is the probability of relevance for a fully mapped attribute. Considering that the probability of setting a GABIL value to 1 is p and the probability of setting it to 0 is $1 - p$, then the probability of generating all possible strings for the binary representation are:

$$\begin{aligned} P(01) &= P(10) = p(1 - p) && \rightarrow \text{String 1 or 0} \\ P(11) &= p^2 && \rightarrow \text{Don't care string} \\ P(00) &= (1 - p)^2 && \rightarrow \text{Unmatchable string} \end{aligned} \quad (6.3.3)$$

Considering this, the probability of relevance of fully and partially mapped attributes would be:

$$r_d^f = l_d P(01 \vee 10) = 2l_d p(1 - p) \quad (6.3.4)$$

$$r_d^p = l_d P(01) = l_d p(1 - p) \quad (6.3.5)$$

Moreover, the last term in Equation (6.3.2) avoids the string 00 in the $d - k$ attributes that are not relevant. If one of the attributes has this string the classifier would not match any instance, and consequently a representative would not be formed. Considering this we can derive the probability of having a representative without using covering or default rule mechanism as shown in Equation (6.3.6).

$$P(\text{rep}) = \frac{2^{k_f} (l_d p(1 - p))^k (1 - l_d (1 - p)^2)^{d-k}}{n} \quad (6.3.6)$$

6.3.1.2 Default class case (no covering)

When there is a default class, there are only $n - 1$ possible actions for a new classifier. Including this assumption, the probability of a representative in this case would be:

$$P(rep) = \frac{2^{k_f} (l_d p (1 - p))^k (1 - l_d (1 - p)^2)^{d-k}}{n - 1} \quad (6.3.7)$$

6.3.1.3 Covering case (no default rule)

To model the usage of covering we need to consider four new aspects over Equation (6.3.2):

1. The number of possible actions will be equal to 1. Since the action will be copied from the example there is no choice besides using that action.
2. The probability of having a string 01 or 10 will depend on the probability of the attribute in the instance being 0 or 1.
3. Using covering having an attribute 00 is not possible anymore.
4. Despite the fact that the classes might be imbalanced the covering opportunities for all the classes are the same.

Considering that we are copying one value from a randomly sampled instance the values for r_d^p and r_d^f will be the same as shown in (6.3.8).

$$r_d^p = r_d^f = l_d(1 - p) \quad (6.3.8)$$

The probability for fully mapped attributes being relevant in this case is reduced by half. This is because since we are copying an instance we could only either generate the string corresponding to the instance value with probability $1 - p$ or the string 11 with probability p .

However, the whole probability of generating a representative in this case depends on the relation of classes mapped by the problem terms m over the total number of classes n . This is because all the classes have the same probability of being selected for covering, but only the classes represented in the schemata will produce representatives. Substituting this assumptions we obtain the probability of having a representative using covering as shown in Equation (6.3.10).

$$P(rep) = \frac{m}{n} (l_d P(01))^k \quad (6.3.9)$$

$$= \frac{m}{n} (l_d (1 - p))^k \quad (6.3.10)$$

6.3.1.4 Covering and default class case

The usage of a default rule mechanisms limits the number of classes that can be used for covering to $n - 1$. Also the number of mapped classes m would not include the default class. Including these changes in Equation (6.3.10) we obtain the probability of generating a representative using covering and default class:

$$P(rep) = \frac{m}{n - 1} (l_d (1 - p))^k \quad (6.3.11)$$

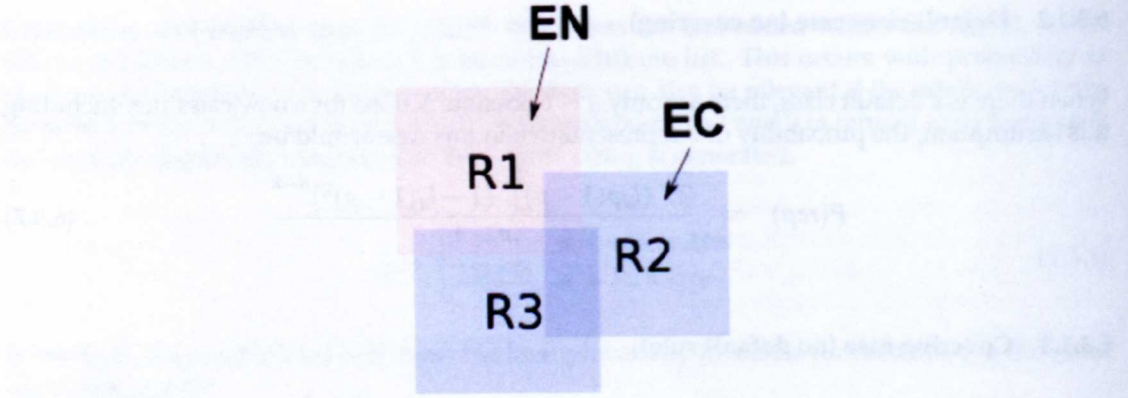


Figure 6.1: Percentage of examples covered by a rule in relation with the total percentage of examples covered

6.3.1.5 Modelling the impact of rule overlap

A randomly generated k-DNF problem (as those used in Chapter 5 to evaluate BioHEL's performance) has a very high probability that some of its terms overlap among themselves, specially if the number of terms r is very high and the number of specified attributes k is small. In this sense, the multiplexer problem is an extremely rare case of k-DNF since none of the terms overlap with each other. In this section, we will extend our models to consider the overlapping between rules.

To calculate this probability we first have to calculate the probability that a rule belongs to a specific niche $P(niche)$. For a problem with no overlapping this probability consists in dividing the probability of a representative by the number of niches or terms.

$$P(niche)_{no} = \frac{P(rep)}{r} \quad (6.3.12)$$

When there is overlapping each term covers the same amount of examples as before. However, the covered space shrinks because there are instances covered by more than one term. This relation is exemplified in Figure 6.1. Considering this, the percentage of space covered by one niche can be generalised by dividing the amount of examples covered by a niche EN among the total number of examples covered EC .

For a randomly generated k-DNF problem with d representable attributes and 2^d instances we know the probabilistic estimation of the amount of positive examples (examples covered) would correspond to:

$$EC = 2^d (1 - P(neg)) \quad (6.3.13)$$

where $P(neg)$ is the probability of generating a negative example. Using the derivation of $P(neg)$ shown in Chapter 5 we obtain:

$$EC = 2^d \left(1 - (1 - 2^{-k})^r \right) \quad (6.3.14)$$

Moreover, we know the amount of examples covered by each rule is:

$$EN = 2^d / 2^k \quad (6.3.15)$$

Substituting the value $1/r$ with EN/EC we obtain that the probability of having a representative

of certain niche when there is overlapping is:

$$P(niche) = \frac{P(rep)}{2^k (1 - (1 - 2^{-k})^r)} \quad (6.3.16)$$

Moreover, we can calculate the probability of having any representative under overlapping conditions $P'(rep)$ by forcing that at least one of the niches has a representative.

$$P'(rep) = (1 - (1 - P(niche))^r) \quad (6.3.17)$$

Particularly in the IRL approach, there is no need that all the niches are represented, as we are interested in learning one rule at the time. Moreover, the previous model is based on the assumption that the rules of a problem cover random subsets of the problem attributes. If the distribution of niches is not uniform, the model does not fully hold.

6.3.2 Covering bound: Probability of matching an instance

To calculate the covering bound we need to calculate the probability of matching an instance with a randomly generated classifier. Although, this was already calculated for the ternary alphabet as shown in Section 6.2.1, in this section we adapt it to the ALKR+GABIL representation. In the following sections we will calculate this probability using and not using the covering mechanism. The default class mechanism in this case does not affect the results obtained in terms of matching. Therefore, no specific formulas are presented for these cases.

6.3.2.1 Base case (No covering)

There are three ways in which a randomly generated attribute can match an example:

- (a) the attribute does not appear in the list with probability $1 - l_d$
- (b) the attribute does appear in the list and the bit corresponding to the instance value is on.

Case (b) occurs with probability p of setting the right bit on. Therefore, the probability of matching is:

$$P(match) = (1 - l_d + l_d p)^d \quad (6.3.18)$$

6.3.2.2 Covering Case

The usage of covering affects the probabilities of setting the right bit on. Therefore, we need to consider two cases:

- (a) the instance used for covering is similar to the one we want to match or
- (b) the instance used for covering is different.

If the instance is similar the probability of matching is 1 and if the instance is different the probability is p , because we need to activate the bit that was not copied through covering. Then the probability of setting the right bit on considering these two cases is $(1 + p)/2$. Substituting p

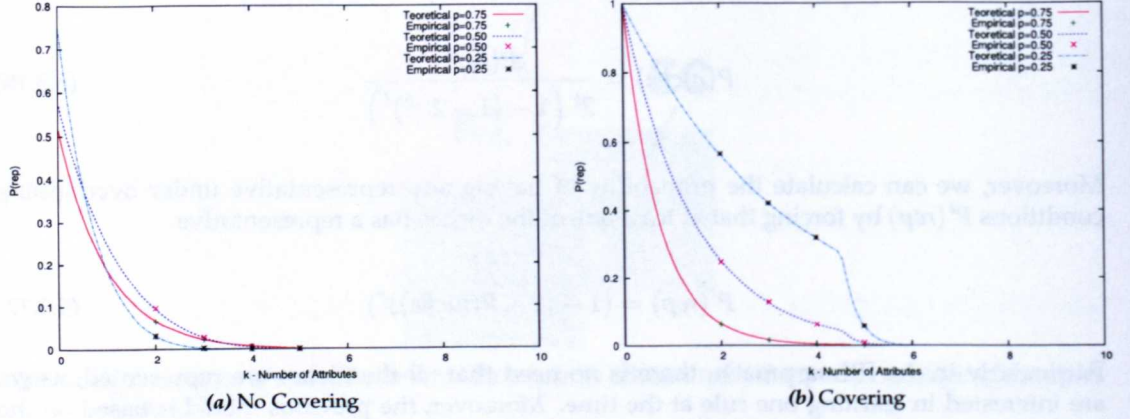


Figure 6.2: Validation of the probability of a representative using the multiplexer problem.

with $(1+p)/2$ on Equation (6.3.18) we obtain:

$$P(\text{match}) = \left(1 - l_d + l_d \left(\frac{1+p}{2} \right) \right)^d \quad (6.3.19)$$

For the worst case, matching an instance attribute when the example used for covering is different happens with probability $1/2$. If the number of instances is smaller than the number of classifiers this factor might grow up to 1, because it is more probable that instances will be repeated during the covering process.

6.3.3 Model validation

To validate the previous models k-DNF and multiplexer problems are used. The k-DNF problems used are of size $d = 10$, with $k = \{2 - 10\}$ and $r = \{1, 5, 10, 20, 40\}$. On the other hand, the multiplexer problems used were of size 3, 6, 11 and 20. It is worth mentioning that there is a direct relationship in the multiplexer problem between the size of the problem d and the number of relevant attributes k as follows: $d = k - 1 + 2^{k-1}$. For the k-DNF problems we generated 5 different instances of each configuration and run each one of them with 25 different seeds. In the case of the multiplexer problems we run each one of them with 125 seeds.² For each run we generated a random population of 500 individuals and calculated different metrics. To validate the probability of a representative we calculated the average number of classifiers that had the bits in the schema represented. On the other hand, to validate the probability of match we calculated the average number of classifiers that match each one of the instances in the problem.

We tested the system under the 4 different combinations of covering and default class mentioned before. For all experiments the value for the *ExpAtts* parameter was 15. Moreover, for the probability p we experimented with three values $p = \{0.25, 0.5, 0.75\}$.

In the following sections we will first validate the probability of finding a representative without overlapping. Afterwards, we will validate the overlapping case and the probability of matching a randomly selected instance.

²We used 125 seeds to obtain the same number of samples as with the k-DNF problems.

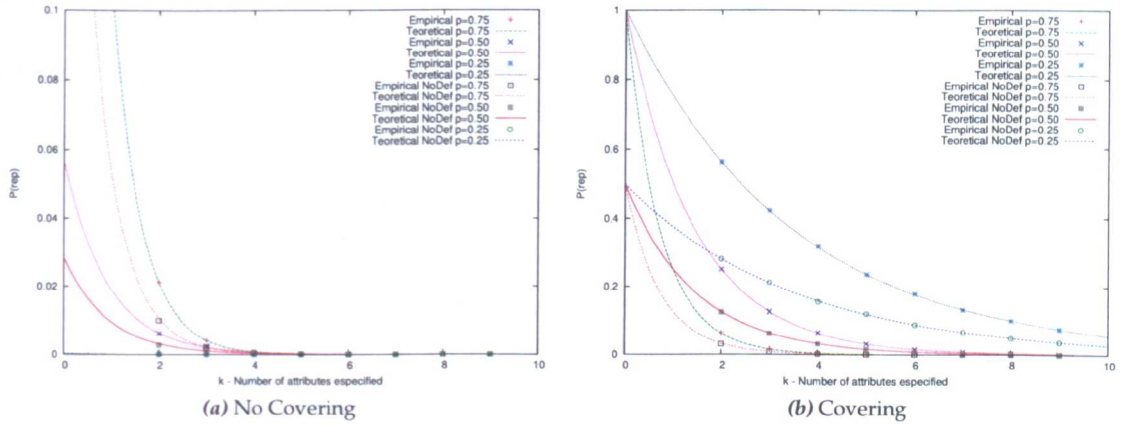


Figure 6.3: Validation of the probability of a representative using the k -DNF problem with one rule

6.3.3.1 Probability of generating a representative

Figures 6.2 and 6.3 show the validation of the probability of finding a representative for non-overlapping problems using the multiplexer and the k -DNF problems, respectively. For the k -DNF models we only used problems with one rule to guarantee that there is no term overlap. In these figures we can see that the models adjust to the empirical data. Moreover, while the number of relevant attributes k increases the probability of having a representative decreases. In particular for the multiplexer problem (Figure 6.2) the probabilities using and not using the default rule mechanism are the same since these problems have schemata that map to all the classes. Therefore, only the results for the covering and not covering cases are shown.

Also it is noticeable in these figures that the mechanisms of default rule and covering increase the chances of creating representatives, which demonstrates the benefits of these techniques. However, the best value of p changes depending on whether the system is using covering or not. When the system does not use covering larger or medium values of p are beneficial. However, when using covering a large p decreases the probability of having a representative, as it raises the chances that a rule misclassifies because it is too general.

6.3.3.2 Probability of generating a representative with niche overlap

To validate the models for problems with niche overlap the k -DNF problems with more than one term are used. Figures 6.4 and 6.5 show the validation of the models considering niche overlap for $p=0.75$ and $p=0.25$, respectively. In these figures we can observe that the models fit the empirical data. Moreover, the probability of generating a representative increases with the number of terms, as there is a higher chance that one of the terms is represented. Also we can observe the benefits of the usage of covering and default rule mechanisms, both of which help increase the probability of obtaining good rules in an initial population. This is particularly noticeable in Figure 6.5 where the usage of covering greatly increases the probability of obtaining individuals that do not make mistakes.

6.3.3.3 Probability of matching a random instance

Figure 6.6 shows the validation for the probability of match using the multiplexer problem. In this case the k -DNF problems were not used because all of them had the same number of attributes $d=10$. We can see here that while the problem size increases the probability of match decreases. Moreover, the higher the value of p the higher the probability of matching an

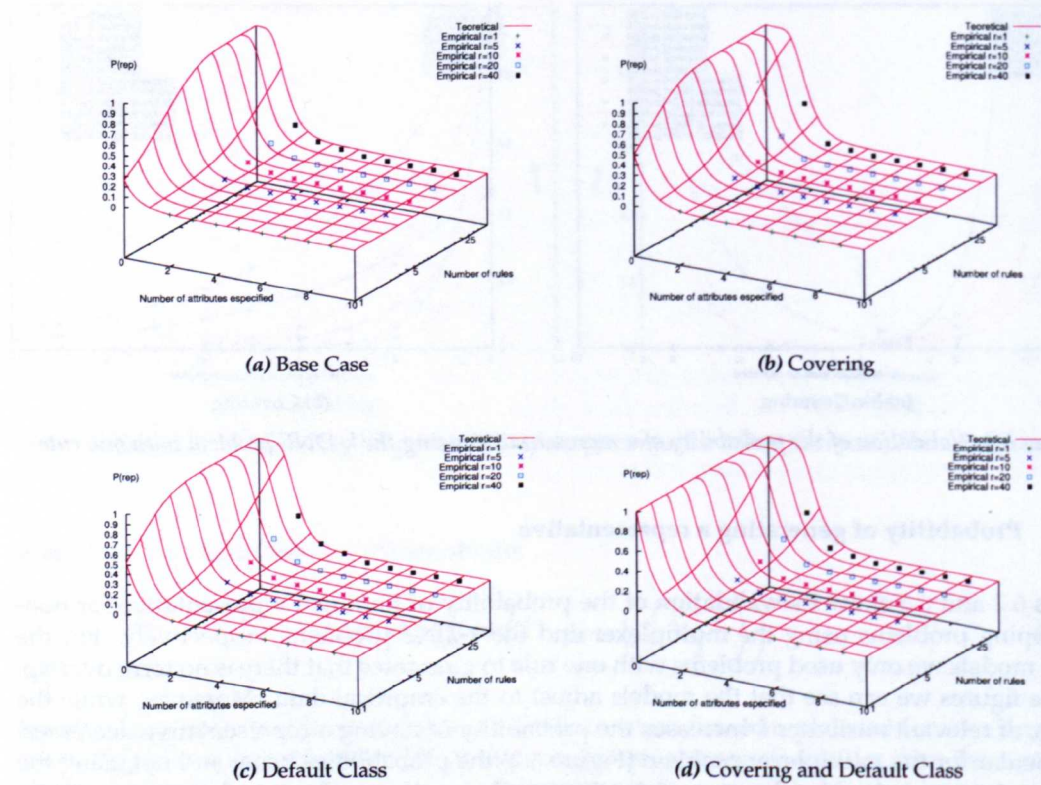


Figure 6.4: Validation of the probability of a representative for k -DNF problem with rule overlap and $p = 0.75$. The graphics show the probability of having a representative considering all the niches

instance since the rules will be more general. The adequate value of p is the largest possible no matter if the system is using covering or not.

Moreover, we can see that when the problem is very small the empirical probability is higher, this is because the problem has less instances than classifiers in the population, so the probability of generating classifiers with the same instance increases. We can also observe a plateau in the models when the number of dimensions in the problem is larger than $ExpAtts$, as attributes that do not appear in the ALKR's list are considered as irrelevant, hence matching any value.

6.4 Towards generalised models for χ -ary attributes

Having derived suitable models for the binary representation in BioHEL we now focus on generalising these formulas to work with nominal attributes with more than two values. This generalisation will show how the problem becomes more difficult when the cardinality of each attribute increases. Also it will show how the GABIL representation gets more robust as the number of values per attribute increase, since this decreases the probability of generating unmatched rules (rules where an attribute has all the bits set to 0). This is a particular issue that has been identified in the literature as a weakness of this representation [Llorà et al., 2007]. In the following sections, we present the generalised formulas for the schema and covering bound for χ -ary attributes.

6.4.1 Schema bound

Let's call the number of possible values of an attribute t and the number of values that need to be set to 1 in an attribute e . For example, for the binary representation considered previously the corresponding values would be $t = 2$ and $e = 1$. In the χ -ary representation with $t = 4$ and $e = 2$ a possible GABIL string would look like:

F1	F2	F3	F4
0101	1100	0011	1001

A representative will be created then if we set to 1 e bits with probability p^e and to 0 the rest of $t - e$ bits with probability $(1 - p)^{t-e}$. Considering this, the formula for the base case can be generalised as follows:

$$P(rep) = \frac{t^{k_f} (l_d p^e (1 - p)^{t-e})^k (1 - l_d (1 - p)^t)^{d-k}}{n} \quad (6.4.1)$$

where t^{k_f} correspond to the number of possible strings that can be generated for fully map attributes.

In this formula, we can see that the probability of creating unmatchable rules decreases while t increases, because the probability of having 0 in all the positions of a GABIL attribute is very small. Figure 6.7 shows how the probability of generating a matchable rule

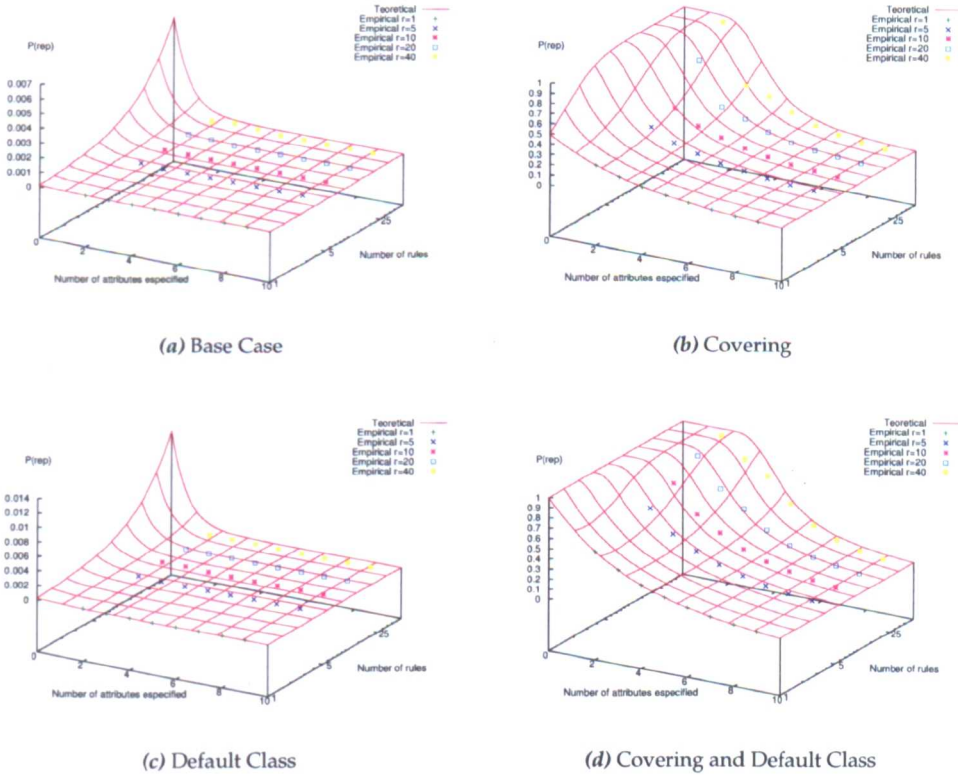


Figure 6.5: Validation of the probability of a representative for k -DNF problem with rule overlap and $p = 0.25$. The graphics show the probability of having a representative considering all the niches

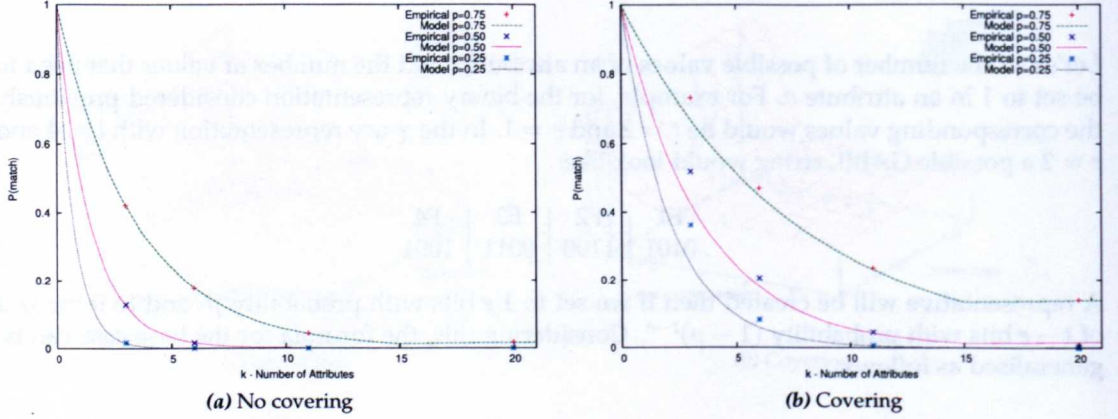


Figure 6.6: Validation of the probability of match using the multiplexer problems.

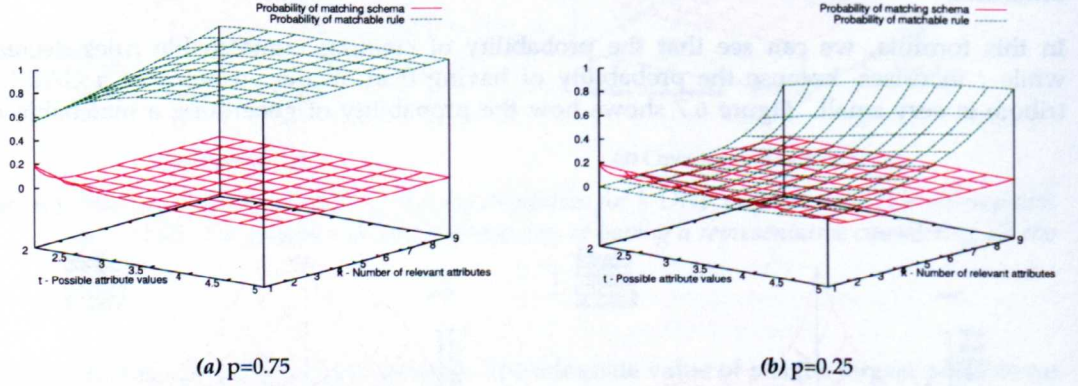


Figure 6.7: Probability of generating the schema attributes against the probability of generating a matchable rule with $e = 1$ and $d = 10$

$(l_d(1 - (1 - p)^t))^{d-k}$ compares to the probability of generating the right string $(l_d p^e (1 - p)^{t-e})^k$ with $e=1$. We can see that while t and k increase it is less likely that we generate rules that do not match. However, the probability of generating the right string decreases when t and k increase. Moreover, as expected the usage of a small p without using covering lowers the probability of generating a matchable rule.

Furthermore, the probability for the default class case is similar to Equation (6.4.1) but subtracting 1 to the number of classes as shown in Equation (6.4.2).

$$P(rep) = \frac{t^{k_f} (l_d p^e (1 - p)^{t-e})^k (1 - l_d(1 - p)^t)^{d-k}}{n - 1} \quad (6.4.2)$$

On the other hand, the usage of covering can be generalised from Equation (6.3.10) by assuming that we copied one bit per attribute from a random instance but there are still $e - 1$ bits to set to 1. The right string will be created then if we put $t - e$ bits in 0 and the other $e - 1$ bits in 1.

$$P(rep) = \frac{m}{n} \left(l_d p^{e-1} (1-p)^{t-e} \right)^k \quad (6.4.3)$$

Similar as the case with no covering, the introduction of the default class only restricts the classifiers considered for covering. Therefore, to calculate the probability for this case we only need to substitute m/n for $m/n-1$ in (6.4.4).

$$P(rep) = \frac{m}{n-1} \left(l_d p^{e-1} (1-p)^{t-e} \right)^k \quad (6.4.4)$$

Moreover, to handle the overlapping case we need to consider that the probability of generating a negative example in the χ -ary representation is:

$$P(neg) = \left(1 - \left(\frac{e}{t} \right)^k \right)^r \quad (6.4.5)$$

where $(e/t)^k$ is the percentage of examples covered by one term of the problem. To summarise, for the overlapping case $P(niche)$ will be equal to:

$$P(niche) = \frac{e^k P(rep)}{t^k \left(1 - \left(\frac{e}{t} \right)^k \right)^r} \quad (6.4.6)$$

6.4.2 Covering bound

When no covering is used the probability of matching an instance is the same as Equation 6.3.18, as it does not depend on the number of values an attribute can accept. Simply with probability p the right bit (the bit corresponding to the instance) will be set to 1. However, when the covering mechanism is activated the probability of generating a matching individual varies because one bit was already set according to a randomly sampled instance. Therefore, this probability will be:

$$P(match) = \left(1 - l_d + l_d \left(\frac{1 + (t-1)p}{t} \right) \right)^d \quad (6.4.7)$$

This means that if the attribute is selected to be in the list we face t possible cases. In one of the t cases the instance used for covering is similar to the one that we want to match. Then the probability of matching is 1. The following $t-1$ cases refers to the cases where both instances (the instance used for covering and the instance we want to match) are different and the rule will match with probability p .

6.4.3 Model validation

To validate these extended models we generated ternary multiplexer problems, which instead of only accepting values 0 and 1 in the attributes they accept an extra value. In this case the size of the problem string would be $d = k - 1 + 3^{k-1}$, where k is the number of relevant attributes in the schemata. We generated ternary multiplexer problems of size 4 and 11. We generated initial

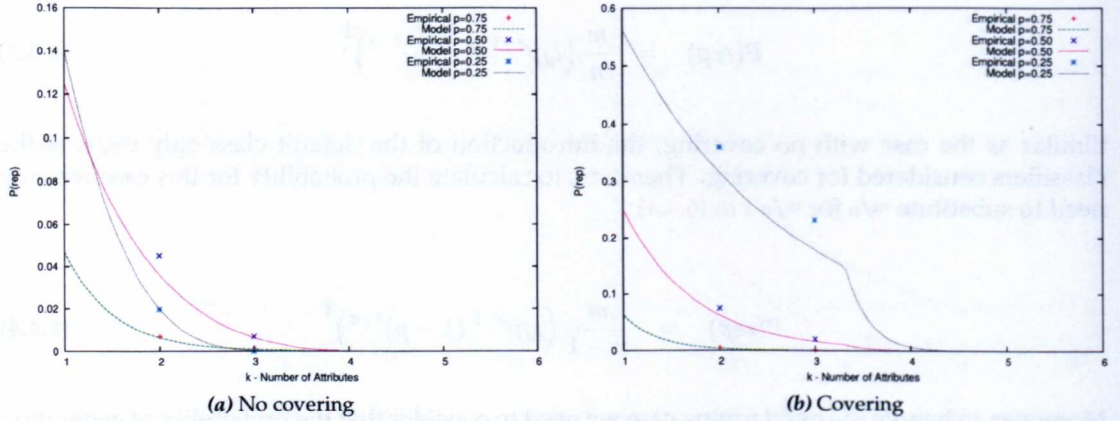


Figure 6.8: Validation of the probability of a representative using the ternary multiplexer problem.

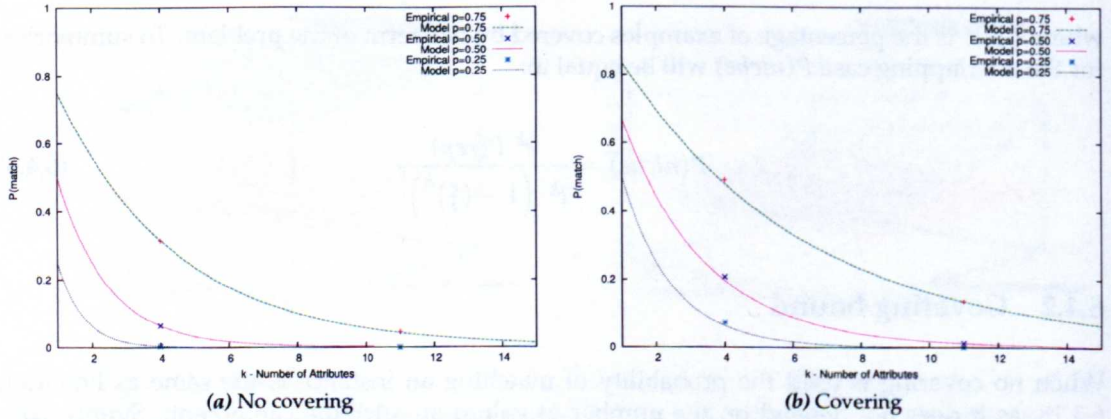


Figure 6.9: Validation of the probability of match using the ternary multiplexer problems

populations with 25 different seeds for each one of these problems and calculated the average percentage of representatives and the average percentage of rules that cover an instance of the training set. Again, the population size was 500 and $ExpAtts$ was 15.

Figure 6.8 shows the validation for the generalised models for the probability of a representative. Since the multiplexer does not show any differences on the probabilities using the default rule, these cases are not shown. In these figures we can see that the models fit the empirical data. However, there is a subestimation in the models because the empirical measures do not filter the cases where the attributes corresponding to the multiplexer address map to two values and the model does. We can notice also that while the domains of the attributes expand the probabilities of generating a representative are lower. Moreover, while a smaller p is better when using covering, $p = 0.50$ gives the best results in the non-covering case, as expected.

Figure 6.9 shows the validation of the generalised models for the probability of match. We can see that in this case the models also fit the empirical data. Moreover, we can notice that the increase of t , the attribute domain, reduces the probabilities of matching compared to Figure 6.6. However, the covering mechanism, as it was shown before, slightly increases the chances of covering the whole search space.

6.5 Ensuring a good initial population in BioHEL

Once derived the probabilities of obtaining a representative and matching a random instance, it is possible to calculate the probabilities of ensuring the existence of a representative and matching the whole search space, which will actually determine the schema and covering bounds for BioHEL.

As shown in Section 6.2.1 the probability of ensuring the existence of a representative or *schema supply* $P(rep\ exists)$ is equal to the complement of not finding any representatives in a population of size N as follows:

$$P(rep\ exists) = 1 - (1 - P(rep))^N \quad (6.5.1)$$

For this formula we can derive the population size bound to ensure the existence of representatives or schema supply with a particular probability $P(rep\ exists) = 1 - \alpha$ (where α represents the confidence interval) as follows:

$$N > \frac{\log(1 - P(rep\ exists))}{\log(1 - P(rep))} \quad (6.5.2)$$

In Figure 6.10 we show the population size boundary with $P(rep\ exists) = 0.95$ and the whole landscape for the schema supply ($P(rep\ exists)$) according to N and p , for binary problems with $k = \{1, 3, 5, 7\}$, $r = 10$ and $d = 10$. In this figure we can observe that for the cases without covering the probability of obtaining representatives grows proportionally to N and p , until $p = 0.6$ where the probability starts dropping again. Moreover, this probability can be very small for large values of k and therefore, the necessary population size to ensure schema supply is really large. In the cases that use covering the situation is different, while the probability of the schema supply still grows proportionally to the population size N , it decreases with p . The larger the problem k the decrease is more steep since the system is interested in generated specific rules and not general ones. The consequence of this is that the necessary population to ensure good individuals increases exponentially while the p increases.

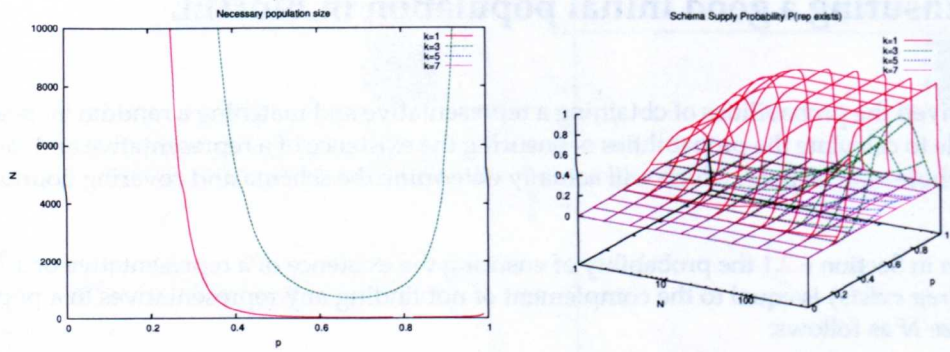
Moreover, the probability that a randomly initialised population of size N covers a particular instance in the training set is equal to the complement of no rule in the population covering that example, as follows:

$$P(cover) = 1 - (1 - P(match))^N \quad (6.5.3)$$

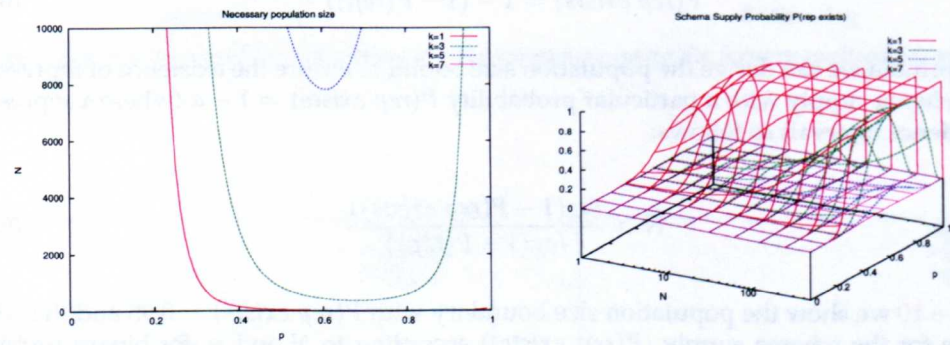
Similar to Equation (6.5.2) we can derive the covering bound for the population size as follows:

$$N > \frac{\log(1 - P(cover))}{\log(1 - P(match))} \quad (6.5.4)$$

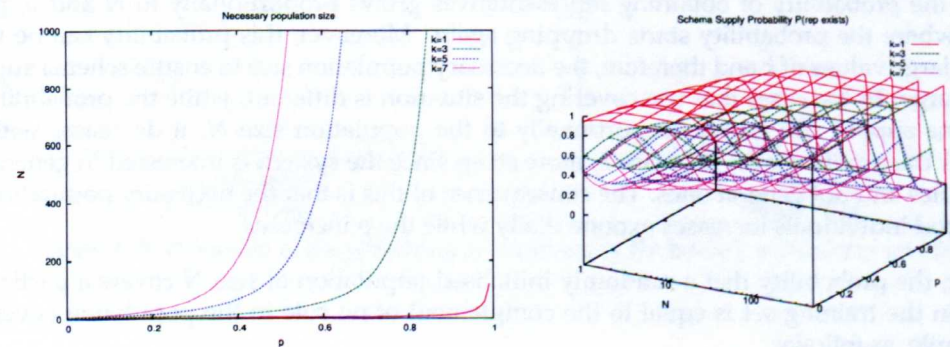
In Figure 6.11 the population size bound is shown assuming $P(cover) = 0.95$ and the whole landscape of the probability of covering $P(cover)$ for problems with $d = \{10, 20, 40\}$ according to N and p . In this figure we can observe that the probability of covering increases when p and N increase. However, as expected, when the system is using covering the probability of covering the search space is higher, and even using lower values of p covering can be ensured if the population is high enough. In this figure we can observe that the necessary population size to ensure covering follows a totally different pattern than the one observed for the schema bound. In this case the population size increases exponentially while p decreases. Other interesting



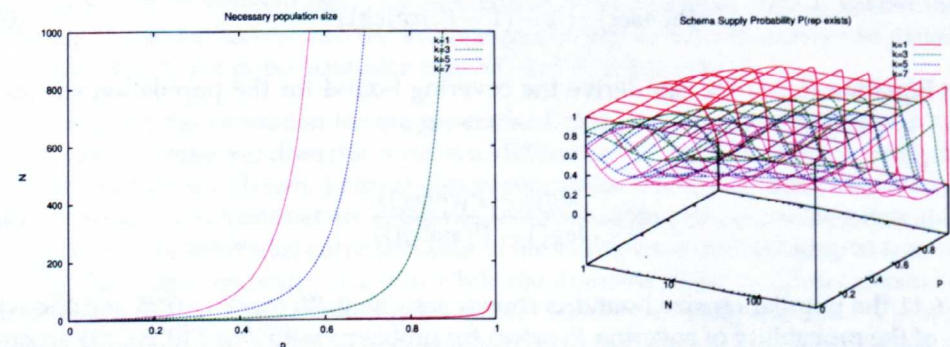
(a) Base Case



(b) Default Class



(c) Covering



(d) Covering and Default Class

Figure 6.10: Population size necessary to ensure $P(\text{rep exists}) > 0.95$ (left) and schema bound probabilities (right) for the different BioHEL configurations in a binary domain $t = 2$, $e = 1$, $d = 10$ and $r = 10$. While p increases the population size needed to obtain representatives increases.

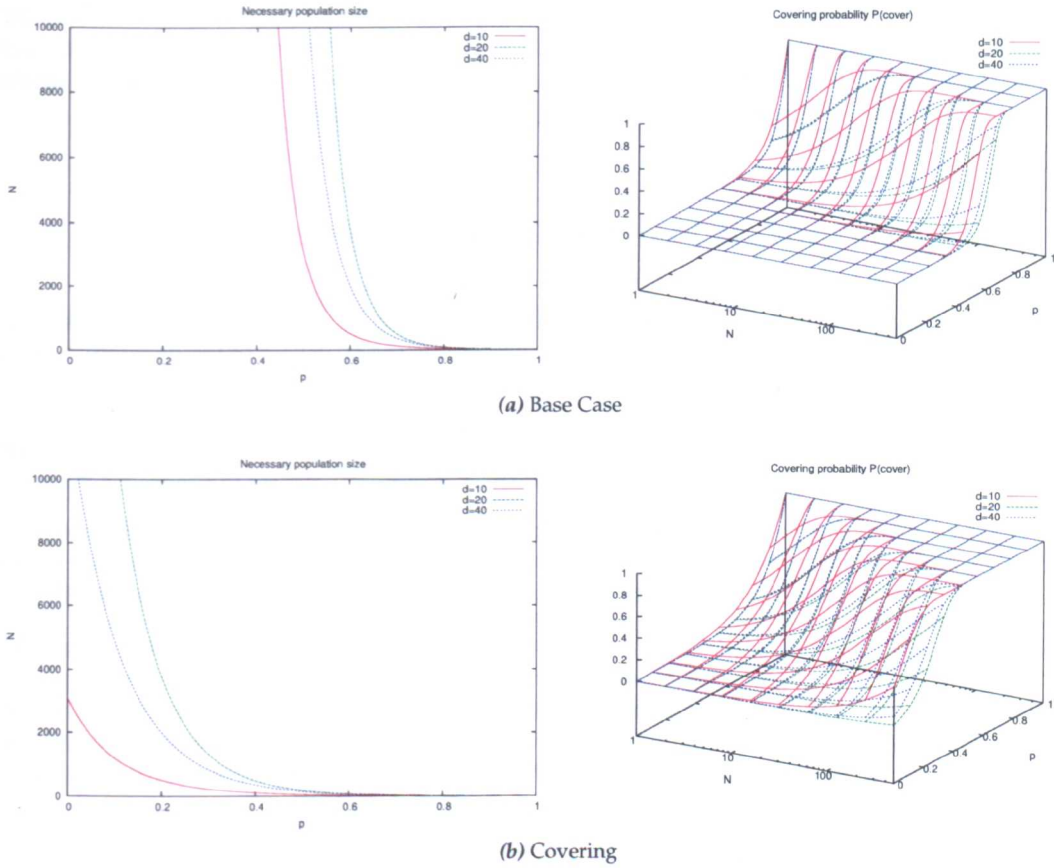


Figure 6.11: Population size necessary to ensure $P(\text{cover}) > 0.95$ (left) and covering probabilities (right) for the different BioHEL configurations in a binary domain $t = 2 e = 1$. While p increases the population size needed to obtain representatives decreases.

thing to notice is that when the problem has 40 attributes the necessary populations are smaller than when the problem has 20 attributes. This is because in ALKR the rules represent on average only 15 attributes and while the problem gets larger, there is a higher probability that the attributes are not selected to be on the list and therefore, they will match any instance.

Considering both bounds it could be possible to determine the minimum population size and the adequate value of p that ensures theoretically the existence of a good initial population in BioHEL, equivalent to the work done for XCS [Butz, 2006]. Figure 6.12 shows the minimum population size needed for problems with $k = \{3, 5\}$, $r = 10$ and $d = \{10, 20\}$. In this figure the point where the two lines intersect can be considered the minimum population size that can be used and the optimal value of p . In this figure we can observe that theoretically as k increases the minimum population size increases and, as expected, the optimal value of p decreases. Moreover, it is worth mentioning that theoretically the population sizes used in BioHEL are really small compared to the ones needed for XCS in [Butz, 2006]. In this figure we can observe that for a problem with 2^{10} examples populations of less than 100 classifiers are enough while for a problem with 2^{20} examples, less than 300 classifiers are enough. On the other hand for XCS the minimum populations necessary are of 3 or 4 orders of magnitude larger. This is due to the fact that IRL learns the problem by learning each niche independently. Therefore, it is not necessary to have all niches represented in the population at the same time, reducing considerably the population resources needed to solve a problem.

Regarding the impact of t and e over the models in the χ -ary representation Figure 6.13 shows the schema supply probability and the covering probability when using covering and default rule, $N = 500$ and $p = 0.5$. Regarding the schema supply we can see that while k increases the

values of t and e that allow a high probability of supply are very restricted and around the lower values. Moreover, it is noticeable that while e tends to t the probability of having representatives increase. Regarding the covering probability, while t increases covering the whole search space becomes more difficult. Moreover, the covering probability does not depend on e .

6.6 Using the models to determine the problem structure

In the conclusions of Chapter 5 we established that in order to set the coverage breakpoint accordingly for binary problems it is necessary to determine the structure of the problem, which is defined with the number of attributes k and the number of terms r . It is noticeable that the models of the probability of obtaining representatives presented in this chapter are dependant on these two values. Therefore, they could be used to derive the structure of the problem if the probabilities of generating a good rule are known beforehand.

However, two problems arise at this point. First, the probability of generating a representative for a problem for which we do not know its characteristics is unknown and we could only estimate this value based on empirical observations. Second, even if we have an estimate for the probability $P(rep)$ these models will only provide a relationship between the k and r values.

The next chapter is focused on overcoming these problems and combine the estimates given by this probabilistic models and the probability of negative examples shown in Chapter 5 to determine the structure of the problem and set the coverage breakpoint accordingly.

6.7 Conclusions and further work

The models presented in this chapter predict satisfactorily the probabilities of generating a good initial population in terms of: (a) covering the search space and (b) generating accurate representatives for each niche in a problem. The models were generated considering the ALKR representation with GABIL encoding. The models show how the problem becomes more difficult depending on the problem characteristics k and r . Moreover, the right selection of the value of p , the probability of setting bits to 1 in the GABIL representation, depends on the initialisation mechanisms uses. While a larger p is beneficial when covering is not used, a smaller p is

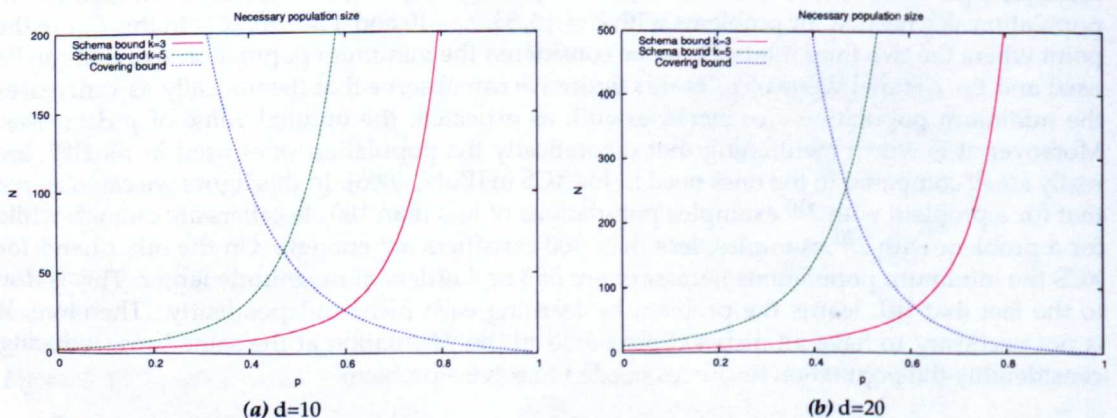


Figure 6.12: Minimum population size to ensure a good initial population according to p with problems with $k = \{3, 5\}$, $d = \{10, 20\}$ and $r = 10$.

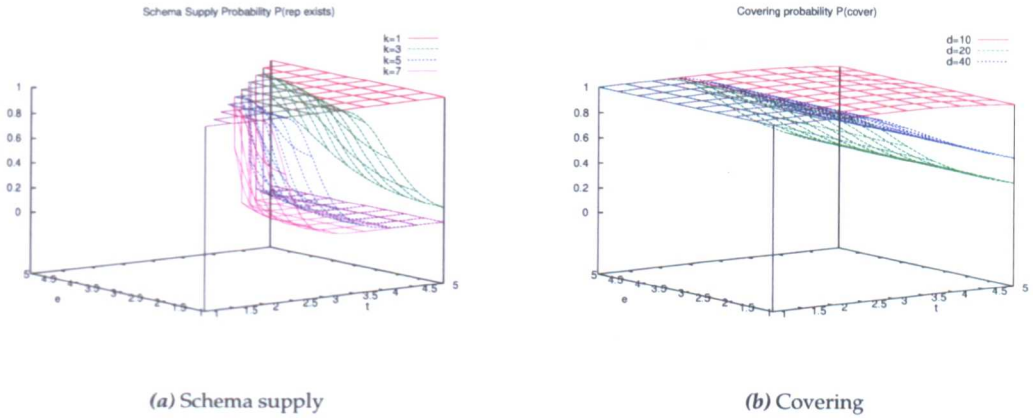


Figure 6.13: Impact of t and e in the schema supply probability for $d = 10$ and $r = 10$ for BioHEL when using covering and default rule.

more adequate when using covering to avoid having a population that is too general.

Moreover, we presented a generalisation of the models for χ -ary attributes. This generalisation showed how the GABIL representation becomes more robust with the increase of the cardinality of the attributes, in terms of generating matchable rules. However, the overall probability of having a good initial population decreases when the number of values per attribute increases, since it is more difficult to generate the right string.

The models overall also show how the covering and default mechanisms introduced in BioHEL improve the chances of generating a better and more useful initial population for the system. Moreover, we show theoretically the minimum population sizes necessary to ensure a good initial population and the optimal values of p depending on the characteristics of the problem. Finally, since the models depend on the problem structure they can possibly be used to determine this structure.

6.8 Further work

Based on these models we proposed a method to automatically determine the structure of the problem and determine the coverage breakpoint accordingly, which will be explained in Chapter 7.

Since we presented models which are based on the assumption that we could estimate several characteristics of the problems. In a further work, we would like to study how can we simplify these models so some of their parameters can be removed without producing a large impact in the accuracy. Moreover, models for the reproductive opportunity, sustenance and learning time should be developed in order to have an unified theory for the correct functioning of BioHEL and, for instance, derive other boundaries for the population size and other user-defined parameters. Furthermore, specific challenges for IRL should be identified in order to model unique aspects of BioHEL's paradigm.

Automatic theory-based adjustment of the coverage breakpoint

Previous chapters have shown that BioHEL has parameters that control its generalisation capacity and setting them right is crucial to achieve a good prediction accuracy. Moreover, it was shown that the adequate parametrisation of the system is strongly related to the characteristics of the problem at hand. As an improvement to the BioHEL system, this chapter presents an automatic parameter setting mechanism that finds the adequate coverage breakpoint value by automatically discovering the structure of the problem. This is done using the theoretical models derived on Chapters 5 and 6. By using these models it is possible to characterise the behaviour of the system when solving a particular problem and therefore determine the problem structure. This method is capable of finding the adequate coverage breakpoint in a wide variety of synthetic and real-world binary problems, improving the system's usability for end-users. Moreover, in the final validation stage our heuristic is able to reduce the computational time of preliminary experiments in up to 71% for a challenging real-world bioinformatics dataset.

7.1 Introduction

EL methods, are well-known for their ability to find solutions to very complex optimisation and classification problems. Their performance, as happens with most methods (both within the context of ML and EAs) is tied to a series of parameters in the system that need to be tuned, and finding the adequate set of parameters can become a very expensive experimental process. Therefore, automatic parameter setting approaches are necessary to avoid a time-consuming preliminary experimentation stage and reduce the number of parameters that need to be set up, making these techniques more accessible for end-users.

Parameter setting is common problem in the field of EL, since these systems use a genetics algorithm and fitness functions that often involve many parameters. Many studies have focused their efforts in finding automatic ways to adapt the parameters in EAs [Eiben et al., 2007b]. Several techniques have been used such as reinforcement learning [Eiben et al., 2007a; Pettinger and Everson, 2002; Sakurai et al., 2010], meta-GAs [Clune et al., 2005; Grefenstette, 1986], self-adaptive approaches [Bäck, 1992; Huy et al., 2009; Krasnogor, 2004; Krasnogor and Gustafson, 2004; Krasnogor and Smith, 2000, 2001; Smith and Fogarty, 1996a,b], among others. In the LCS context most parameter control studies use self-adaptation strategies [Bull and Hurst, 2000; Butz et al., 2008c; Hurst and Bull, 2001a, 2002].

The difficulty mentioned above is also present in BioHEL as it was shown on Chapters 4 and 5, where extensive parameter sensitivity analyses were performed. One of the main characteristics of BioHEL is its fitness function, which tries to balance the accuracy, complexity and coverage of the rules in the solution. The key element of this fitness function is the *coverage breakpoint* parameter, which determines how many examples a rule should cover to be consid-

ered good and general enough. Chapter 5 has shown that the learning can be facilitated when it is correctly set up, but also an incorrect setting can push the system towards overgeneralisation. Moreover, this parameter is very problem dependant and requires an extensive preliminary experimentation to determine its adequate value.

Moreover, as it was shown in previous chapters, it is possible to set the adequate coverage breakpoint for binary functions if the structure of the problem is known. In the case of binary problems, their structure can be characterised using k-DNF boolean formulas, which main characteristics are the number of relevant attributes (k) in each conjunctive term and the number of conjunctive terms (r) in the k-DNF formula. Considering this, the problem of determining the coverage breakpoint can be translated to determining the k and r . However, determining these values is not straightforward and calculating their exact value would involve applying data mining over the problem which is an extra computational cost that we wish to avoid.

This chapter introduces an automated heuristic approach to determine the structure of the problem (k and r) and set the coverage breakpoint parameter accordingly. This heuristic combines observations over the actual data and over a sample of randomly initialised rules evaluated against this data, to retro-feed the theoretical models of the behaviour of the system derived in Chapters 5 and 6, and classify the problems into groups called *kr-groups* with a particular k and r associated.

The results show that the proposed method is capable of characterising challenging binary problems with and without noise. The additional effort incurred by the heuristic, in terms of number of additional evaluation operations, is also analysed and techniques to regulate this effort are discussed.

Moreover, the final test on a real-world protein structure prediction problem showed that the system managed to adapt the coverage breakpoint parameter to the best value found through exhaustive experimentation. Our approach not only reduced the execution time of the whole experiment (including the parameter adjustment phase) in 71%, but also intrinsically adapts other parameters of the system, such as the specificity used to initialise the rules.

The rest of the chapter is organised as follows. Section 7.2 introduces the previous studies around automatic parameter adjustment in EAs and LCS. Section 7.3 describes our approach to determine the problem structure and its corresponding coverage breakpoint value. Section 7.4 introduces the experimental design and results, and finally Section 7.5 summarises the conclusions and further work.

7.2 Parameter setting and parameter control methodologies

Automatic parameter adjustment is a very challenging process in EL and, in general, in EAs, where it has been studied in great length. In this section we first describe, from a general EA perspective, the different approaches to parameter control (an some examples of each). Afterwards, we focus on specific examples of parameter control in EL.

There are different types of parameter control algorithms. Eiben et al. [2007b] presented a classification for parameter control where the different techniques can be classified depending on what is changing (representation, mutation of crossover rates, selection mechanisms, etc.), or depending on how the change is made (deterministic, adaptive and self-adaptive). The *deterministic* techniques are the ones where the parameter changes without having any feedback from the search. The *adaptive* techniques are the ones that use some sort of feedback from the search. The *self-adaptive* techniques are the ones which evolve the parameters along with the rest of features of the problem. Using this classification our approach can be classified as a deterministic one.

Moreover, the parameter control can also be characterised by the evidence used to produce the change: absolute evidence or relative evidence [Eiben et al., 2007b]. While the first one changes a parameter based on a certain event, the latter compares information of the current state with the information from other chromosomes.

In the area of EAs, the first approaches were presented by Rechenberg [1973] and Schwefel [1975] where the mutation rate was adapted according to the *1/5th rule*. If the rate of successful mutation was over $1/5$ the mutation rate was increased and if it was below this value it was decreased. Other early approaches involve the adaptation of crossover rate depending on how good were the resulting offsprings [Davis, 1989; Davis and Mitchell, 1991].

Regarding operator selection within EAs, Spears [1995] presented a very simple self-adaptive crossover selection method. One extra bit in the classifier encoding represented the crossover that should be applied. Other approaches [Carvalho and Araujo, 2009; DaCosta et al., 2008] use rules to modify the parameters of local search operators (crossover and mutation). DaCosta et al. [2008] presented a new adaptation rule that modifies the selection of both crossover and mutation based on how well these operators have performed in terms of improving the fitness. Their implementation is based on Multi-Armed Bandit [Auer et al., 2002], where they added a Page-Hinkley statistical test to detect changes along with time. Afterwards, this study was extended by Fialho et al. [2009] where he compared the four operator selection mechanisms of the previous study with four different credit assignment mechanisms for the operators. Moreover, Carvalho and Araujo [2009] used rules based on the diversity of the population to adapt the mutation rate for the NSGA-II system [Deb et al., 2000], a Multi-Objective Evolutionary Algorithm (MOEA). Using the crowding distance¹ they calculate the diversity in the population and then generate a probability distribution for mutating a value. This probability distribution is dependant on the diversity in a way in which the mutation is strong when the solution are disperse and far from the Pareto optimal front, but soft when the solutions are closer to the Pareto optimal front.

Reinforcement Learning (RL) has also been used widely to adapt parameters in EAs. This means the implementation of an agent that adjust the parameters of the system based on the results obtained. For instance, Müller et al. [2002] applied RL to identify the appropriate step size for the *1/5th rule* when adapting the mutation rate. The results in this case showed that the success of this approach depends greatly on the reward function used for the RL mechanism. Eiben et al. [2007a] also applied a RL approach, but to adapt all the parameters instead of the mutation rate only. In this study the usage the new approach outperforms a benchmark GA, however the number of evaluations needed is larger when using RL. Furthermore, other more complex approaches [Sakurai et al., 2010] use RL to adapt the parameters of the GA considering not only the quality of the solutions, but also the cost incurred by the selected search operators.

Other examples of parameter control involve the usage of meta-GAs to tune parameters within a GA [Clune et al., 2005; Grefenstette, 1986]. In these studies a first GA was used to search within the search space of parameters of a second GA, to determine the parameter combination that works better for a certain problem while solving it.

Specifically in LCS, there are plenty of examples of self-adaptiveness. Bull and Hurst [2000] implemented a self-adaptive mutation in ZCS. The self-adaptive mechanism consisted in giving an independent mutation rate to each classifier, which is stored and evolved within the classifier. These experiments showed that the self-adaptive mutation performs as well as using the optimal parameters. However, this was a prove of concept that it was possible to implement self-adaptive mutation in ZCS and reduce the cost of finding the optimal parameters by trial and error. Afterwards, this work was extended by self-adapting all the parameters in ZCS (mutation, learning rate, tax rate and discount factor) at the same time [Hurst and Bull, 2001b]. While in stationary environments the results were as good as the ones obtained with fixed parameters, in the more dynamical ones the self-adaptation improves the performance of the

¹The crowding distance of a solution is an estimate of the density of solutions surrounding that solution [Deb et al., 2000]

system. They also showed that it was more beneficial to adapt all the parameters at the same time, than just the mutation rate. Afterwards, Hurst and Bull [2002] tried to self-adapt both the mutation and the learning rate in XCS. The adaptive mutation rate solved some generalisation problems on XCS. Nevertheless, the performance was still sub-optimal in this case. Also the system showed worst performance when trying to adapt the learning rate.

Finally, Butz et al. [2008c] also implemented the previous principles of self-adaptive mutation to XCSF using hyper-ellipsoidal condition structures. In the first stage of experiments they adapted only one value which was used to perform mutation uniformly over the different parts of the hyper-ellipsoid (centre, stretch and rotation). Using this first approach there was no improvement in the performance of the algorithm and the only benefit was that the system was capable to learn the problem even with wrong initial mutation settings. Afterwards, they also tested self-adapting individually the mutation rate for the different parts of the hyper-ellipsoids and the results showed an improvement in the accuracy compared to XCSF using fixed parameters.

Other novel and completely different approaches promote coding on top of an abstract framework in which the programmer focuses in developing the algorithms to solve a particular task (i.e. data mining tasks) and the framework performs the algorithmic optimisations and determines the adequate parameter settings [Hoos, 2012]. In this approach the parameter setting is just a small part of the whole meta-algorithmic optimisation performed on top of an abstract coded task. This approach so far has only been tested for solving integer programming, planning and satisfiability problems, obtaining speedups up to 525X in the whole algorithm including the parameter setting stage.

7.3 Automatic parameter setting of the coverage breakpoint

Considering the importance of the coverage breakpoint parameter in the performance of the BioHEL system, it seems necessary to adjust it automatically for several reasons:

Run-time. Since BioHEL is a system mainly oriented to solve large scale datasets, finding the correct setting for problem dependant parameters such as the coverage breakpoint involves a time-consuming preliminary experimentation stage. The automatic setup of this parameter can avoid preliminary experimentation and reduce the total experimental time.

Usability. In many cases end-users avoid exhaustive experimentation and settle for naive configurations that do not produce the best results. This improvement could make the system easier to use by an end-user and could also find better solutions for problems where the adequate coverage breakpoint has not been determined yet.

According to Chapter 5 it is possible to determine the coverage breakpoint if the characteristics of the problem (k -DNF formula) are known. These characteristics are the number of attributes expressed in the terms (k) and the total number of terms in the formula (r). According to this chapter if the number of attributes in the terms is k the adequate coverage breakpoint to solve the problem is 2^{-k} . To ensure learning the coverage breakpoint should be equal or smaller than this value. This translates the problem of finding the adequate coverage breakpoint to finding the k value of the problem.

But how it is possible to determine the structure of the problem by only observing the data? To do this it is necessary to have models based on k and r that explain a) the characteristics of the data and b) the behaviour of the system when working this data. For example, the model for the number of negative examples shown in Chapter 5 finds a correlation between k and r and the proportion of negative examples in the problem. Moreover, the probability of obtaining good individuals in BioHEL shown in Chapter 6, also provides a relationship between a characteristic of the initial population and the variables k and r . This means that by observing these characteristics we can have estimates of the k and r of the problem.

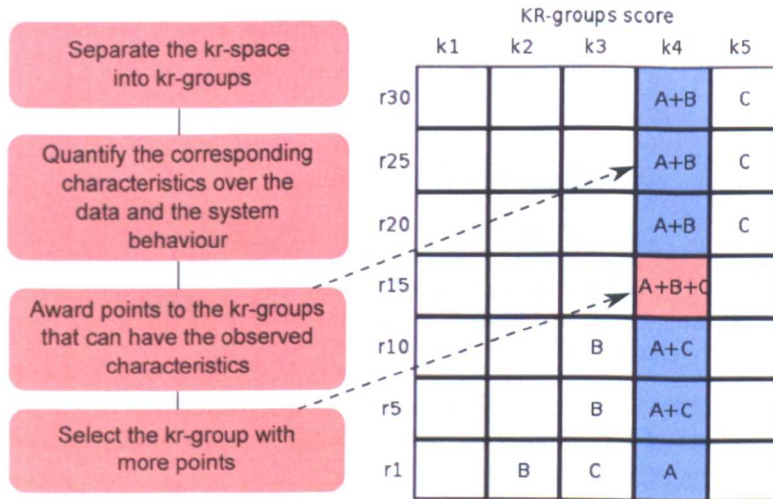


Figure 7.1: General diagram of the usage of the heuristic to determine the structure of the problem.

Since we need to retro-feed the models, each model independently will only indicate a relationship between k and r . Therefore, many different values for k and r can satisfy the equation and individually each model does not give too much information. Nevertheless, by combining the results of different models together it is possible to determine the values for k and r that are more likely to match the characteristics of the problem.

In this chapter we present a heuristic approach to determine the k and r of a given boolean problem at run-time. This approach works by classifying the problems based on observations made over the data and randomly sampled individuals. Using this information the problems are classified into groups with a particular k and r associated, which we will call *kr-groups*. The classification is done using a voting system. The space of k and r is divided uniformly in *kr-groups* which have an associated expected value and standard deviation boundaries for each one of the characteristics we intend to measure. When a problem presents a characteristic that falls into the standard deviation boundaries of a particular *kr-group* the group gets awarded points. At the end the *kr-group* that obtained more votes is considered the winner. Figure 7.1 illustrates how the heuristic works.

Particularly for BioHEL, three characteristics were considered to classify the problems:

- The number of negative examples in the problem.
- The number of *good individuals* in a random sample after evaluating them against the given problem. These are individuals that do not make classification mistakes or have an accuracy higher than a threshold.
- The number of attributes expressed in the good individuals.

The first two characteristics used are completely theory-driven. The last characteristic, even though it does not come from a model, reinforces the two previous criteria in finding the correct *kr-group*. Since the good individuals are already calculated for the second criteria, using the number of attributes expressed in these individuals does not involve an extra computational cost.

7.3.1 Classification criteria

The following sections will explain in greater detail each one of the criteria used to classify the problems. Afterwards, we will show in more detail how the *kr-space* is partitioned in *kr-*

groups and what makes a problem belong to a specific group. Finally, we explain the algorithm step-by-step.

7.3.1.1 Number of negative examples in the problem

The negative examples are defined as the examples covered by the default class (examples that are not going to be classified by the generated rules). Depending on the number of terms r and number of attributes expressed in each term k , the k -DNF problem will present a different percentage of negative examples.

For a randomly generated binary problem defined as the disjunction of r terms, where each term is the conjunction of k randomly picked attributes, the probability of having a negative example in the training set $P(neg)_r^k$ is equal to:

$$P(neg) = \left(1 - \frac{1}{2^k}\right)^r \quad (7.3.1)$$

By counting the number of negative examples in the training set, it is possible to use this formula inversely to determine possible combinations of k and r that are feasible for the given problem. For example a problem with $k = 2$ and $r = 1$ has 75% of negative examples. But also a problem with $k = 6$ and $r = 18$ has on average the same percentage of negative examples. If we observe a particular problem with 75% of negative examples both of these kr -groups would receive scores according to this criterion.

7.3.1.2 Number of good individuals in a randomly initialised sample

A good individual or a *representative*, as explained in Chapter 6, is a classifier that specifies correctly all least all the attributes in one of the terms of the optimal solution to the problem. Therefore, this rule does not make mistakes, but it can be more specific than the optimal rule where only k attributes are specified. The probabilities of finding a representative were first proposed by Butz [2006] for the ternary representation $\{0, 1, \#\}$ and extended in this thesis for the binary domain using the ALKR+GABIL representation in Chapter 6.

Assuming the usage of the default rule and covering mechanisms, the probability of finding a representative in BioHEL for a randomly generated binary problem depends on k and r , and it is equal to Equation (7.3.2). This function states that the probability of having a good classifier $P(rep)$ is equal to the probability of having at least one of the terms in the k -DNF problem represented, and to have a term represented the rule should expressed the k relevant attributes. For more details about this model please see Section 6.3.1.

$$P(rep) = 1 - \left(1 - \left(\frac{2^{-k} (ld(1-p))^k}{1 - (1-2^{-k})^r}\right)\right)^r \quad (7.3.2)$$

In this formula p corresponds the probability of setting to 1 the values in a GABIL attribute, and ld is the probability that an attribute appears in the ALKR attribute list (see Section 2.4.2). This last value depends on the user-defined parameter *ExpAtts* as follows:

$$l_d = \begin{cases} 1 & d \leq ExpAtts \\ \frac{ExpAtts}{d} & d > ExpAtts \end{cases} \quad (7.3.3)$$

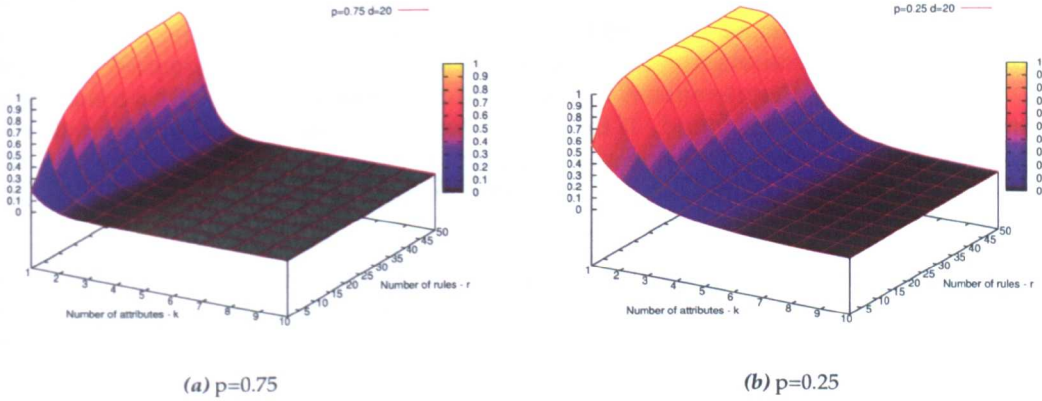


Figure 7.2: Probability of generating a representative with different values of p in a problem with $d = 10$ and $\text{ExpAtts} = 15$.

Figure 7.2 shows an example of the landscape of this model using different values of p . By counting how many representatives are found in a randomly initialised sample of individuals it is possible to use the formula inversely to determine feasible pairs of k and r for the given problem.

The individuals for the sample are not generated one by one, but by chunks of N individuals (for all experiments in this chapter $N=500$). After evaluating N rules, it might be possible that we do not find any representatives. This could happen due to several reasons. Either the sample is too small and/or the probability of a representative for a particular point is too small as well. To solve these problems the system increases iteratively the total sample size (generate more additional samples of size N) until a number of representatives R (parameter set by the user) are found. This guarantees that the number of representatives found is not zero while checking as few individuals as possible. A high value of R will involve checking a bigger sample size, while a small value would have the opposite effect.

Algorithm 7.3.1: SEARCHREPS(N)

```

 $p \leftarrow p_{max}$ 
while  $p \geq p_{min}$ 
   $rep \leftarrow \emptyset$ 
  while  $i < 6 \vee rep \neq \emptyset$ 
    sample  $\leftarrow \text{GENSAMPLE}(N, p)$ 
    for  $c \in \text{sample}$ 
      do {
        if  $\text{GETACCURACY}(c) \geq \text{minAcc}$ 
          then {  $c \leftarrow \text{PRUNING}(c)$ 
                 $rep \leftarrow rep \cup c$ 
              }
        if  $|rep| \geq R$ 
          then  $rep \leftarrow \text{ERASEOUTLIERS}(rep)$ 
         $i = i + 1$ 
      }
     $p = p - p_{step}$ 
  return (null)

```

Moreover, we try to generate R representatives using the largest value of p (probability of setting to 1 the bits in a GABIL string) possible, because that would create more general rules as shown in Figure 7.2. However, more general rules are more likely to make mistakes. Therefore, when the problem has a larger k , smaller values of p are needed to generate rules that do not

make mistakes. The algorithm the system follows to adjust p , while finding the representatives, is shown in Algorithm 7.3.1. The system first tries to obtain representatives generating populations of size N created using the largest value of p : p_{max} . Then all these individuals are evaluated against the training set. Afterwards, all the rules with accuracy higher or equal than $minAcc$ are considered representatives. When R or more representatives are found the system returns the representatives found. If the system has already checked 6 samples and has not found any representatives, the value of p is lowered globally across the system and the search continues. If the value of p has reached its minimum value and the system has not found representatives yet, the search aborts.

The calculation of representatives from a given sample is interesting because it gives room for our third criterion, which is the number of attributes observed in them. However, the calculation of the genuine representatives is not straight forward and some post-processing is needed as it will be explained in the next section.

7.3.1.3 Number of attributes in the good individuals

According to the definition of a representative the number of relevant attributes in a candidate representative cannot be less than k , because otherwise this rule will make mistakes. This gives us at least an upper bound of the k of the problem. However, in order to use the characteristics of the representatives we need to make sure that these "good rules" are actually genuine representatives.

Two problems might arise with the good rules. First, it can happen that the good rules have more attributes specified in the actual terms of the problem, which is misleading. Second, when the problem consists of more than one rule, the system might find classifiers that do not make any mistakes but they do not represent the terms of the problem. These classifiers instead represent the union or intersection between two or more terms. They might have more or less attributes expressed than k , and they are not really representatives of the problem. Therefore, in order to find genuine representatives it is necessary to preprocess them.

To tackle the first problem we need to eliminate the attributes that do not affect the accuracy. This process is called pruning and it tries to generalise the rules as much as possible. To prune unnecessary attributes we perform an iterative search process as shown in Algorithm 7.3.2. Each one of the attributes in the rule is eliminated, one by one. If the accuracy decreases, the classifier is restored, if not the search continues over the resulting classifier. This local search operator can also be used as a post-processing operator to refine the generality of the rules. This will be studied in-depth in Chapter 9.

Algorithm 7.3.2: PRUNING(Classifier $c1$)

```

prevacc ← GETACCURACY( $c1$ )
for each  $att \in$  GETATTRIBUTES( $c1$ )
do {
  REMOVEATTRIBUTE( $att, c1$ )
  if GETACCURACY( $c1$ )  $\geq$  prevacc
  then prevacc ← GETACCURACY( $c1$ )
  else RESTORE( $att, c1$ )
return ( $c1$ )

```

To tackle the second problem we need to eliminate the deceptive representatives. That is, the classifiers that do not make mistakes but do not really correspond to the terms of the problem we want to learn. Since these rules usually have a number of attributes either larger or smaller than k (but not exactly k), we keep only the ones that correspond to the most frequent number

of attributes observed k^* . This routine is exemplified in Algorithm 7.3.3. After erasing the outliers, the rules left in the set are considered genuine representatives and they are the input for the second criterion (as shown in Algorithm 7.3.1). Moreover, as we have already calculated the most frequent number of attributes observed among the representatives, we can also use this information k^* as a third metric to award points to the kr -groups with $k = k^*$.

Algorithm 7.3.3: ERASEOUTLIERS(rep)

```

 $k^* = \text{GETFREQK}(rep)$ 
for  $c \in rep$ 
  do { if  $|c.atts| \neq k^*$ 
      then  $rep = rep - c$ 
    }
return ( $rep$ )

```

7.3.2 How to classify the problems?

As we explained before, to use the model it is necessary to calculate the standard deviation boundaries for each one of the possible combinations of k and r . Towards this aim, we sample uniformly the kr - space and calculate the expected value, the lower bound and upper bound for each point. To sample the space we calculate these values for $k = \{1..d\}$ and $r = \{1..rmax\}$ using a step size for the rules $rstep$. According to our preliminary experimentation $rmax$ should be at least equal to the number of rules we expect to find in the problem.

Moreover, to calculate the lower and upper bounds for each point the probabilistic models presented need to be analysed further. For the probability of a representative we can consider that the probability of having a specific number of representatives in a sample of N classifiers follows a binomial distribution with probability $P(rep)$. This assumption comes from the fact that the generated rules have the same probability of becoming representatives and they are independent from each other. Therefore, the probability of having exactly x representatives in a sample of size N can be written as follows:

$$P(\#rep = x) = \binom{N}{x} (1 - P(rep))^{N-x} (P(rep))^x \quad (7.3.4)$$

In this case we know that for each point the mean percentage of representatives is $P(rep)$ and the variance is $var = P(rep) \cdot (1 - P(rep))$. So for a problem to belong to a specific kr -group, the system has to check if the empirical percentage of representatives observed $P'(rep)$ is between the boundaries as follows:

$$P(rep)_r^k - var \leq P'(rep) \leq P(rep)_r^k + var \quad (7.3.5)$$

In the case of the probability of having a negative example we do not know the probability distribution, but we know the mean value given k and r . We cannot assume that it is a binomial distribution because the examples in a training set are not independent observations (usually they are not repeated). In this case fixed intervals are used to determine if a problem belongs to a certain group. To do this we calculate the empirical value $P'(neg)$ and use a constant α to classify the problems as follows:

$$P(neg)_r^k - \alpha \leq P'(neg) \leq P(neg)_r^k + \alpha \quad (7.3.6)$$

7.3.3 Parameter setting procedure step-by-step

To recapitulate and explain better how the concepts and methods presented are merged together to produce our approach, in this section we will explain step-by-step the algorithm used within BioHEL to determine the k and r of a problem and its corresponding coverage breakpoint. The algorithm consists of the following steps also shown graphically in Figure 7.3.

1. Determining the number of attributes in the problem d and the value ld .
2. Searching R representatives in randomly initialised populations. Finding representatives involves the following sub-steps:
 - (a) **Exhaustive search.** Searching iteratively in populations of N classifiers until a total of R representatives is found.
 - (b) **Representative pruning.** When a good rule is found, the system removes all the attributes that can be eliminated without degrading the accuracy.
 - (c) **Adjustment of initialisation parameters.** If the system have checked already 6 populations and have not yet found any representatives, the system re-adjusts the value p (See Section 7.3.1.2).
3. Calculating the most frequent number of attributes activated in the candidate representatives, erasing the misleading ones. This means keeping only the ones that have a k equal to the most frequent value observed k^* .
4. Determining the number of examples in the training set belonging to the default class ($P'(neg)$).
5. Calculating the observed value $P'(rep)$ as the number of representatives observed divided by the total number of rules observed (total sample size).
6. Calculating the score of a kr -group ($Score_r^k$) with Equation (7.3.7) where $k = k^*$, Neg_r^k and Rep_r^k are boolean variables that take the value of 1 if the empirical observation matches the criteria of the kr -group.

$$Score_r^k = A \cdot (k == k^*) + B \cdot Neg_r^k + C \cdot Rep_r^k \quad (7.3.7)$$

$$Neg_r^k = P(neg)_r^k - \alpha \leq P'(neg) \leq P(neg)_r^k + \alpha \quad (7.3.8)$$

$$Rep_r^k = P(rep)_r^k - var \leq P'(rep) \leq P(rep)_r^k + var \quad (7.3.9)$$

7. To finalise we calculate which is the smallest group (smallest k and r) that obtained the highest coincidences between the 3 metrics (highest score). This k is transformed in the coverage breakpoint as $CB = 2^{-k}$.

7.4 Experimental design and results

This section presents the experimental framework used to test our approach and the corresponding results. First, the parameter setting approach is analysed over a wide variety of synthetic k-DNF problems, with and without noise, created as explained in Section 5.2. Afterwards, an extra test of our approach over a binary protein structure prediction problem *CN-bin*

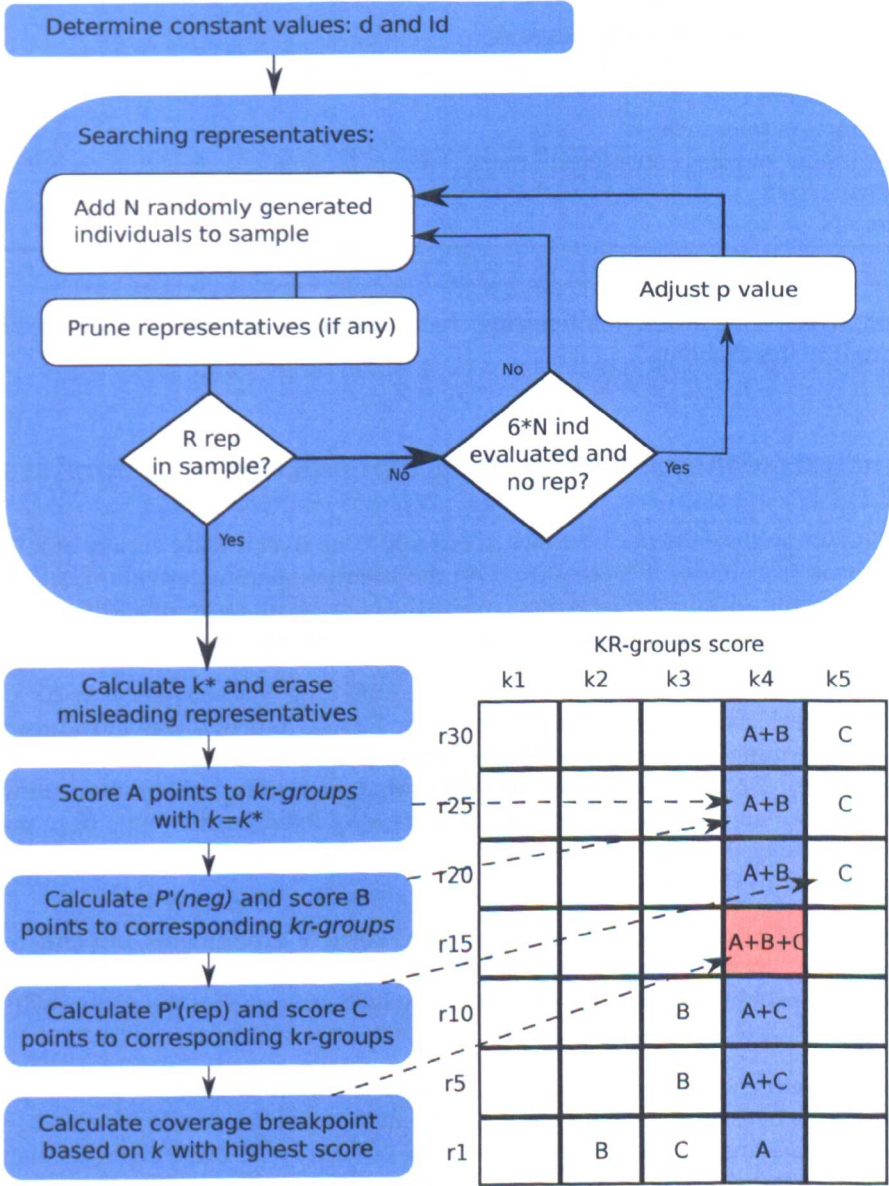


Figure 7.3: Steps to find the adequate coverage breakpoint with and example of the final score grid.

Table 7.1: Parameters for the heuristic used to characterise and find the coverage breakpoint for k -DNF problems.

Parameter	Value
Number of representatives needed - R	10
Evaluated pops to change p	6
Prob p initial value - p_{max}	0.75
Prob p minimum value - p_{min}	0.25
Prob p decreasing step - p_{step}	0.25
Most frequent k in representatives - Score A	2
Imbalance function - Score B	2
Prep function - Score C	1
Imbalance function tolerance - α	0.1
Maximum number of rules considered - r_{max}	100
Rule step size - r_{step}	5
Sample size - N	500

is presented, which constitutes an interesting challenge for our approach due to the high levels of noise found in the problem.

7.4.1 Analysis of the parameter setting approach over binary problems

In this section we analyse the performance of our approach over a wide variety of k -DNF problems in terms of probability of success (finding the adequate parameter value). At the end, we also comment on the additional effort incurred by the heuristic in terms of additional evaluation operations, and how the effort can be regulated by some parameters.

The k -DNF problems used in this section have the following characteristics: $d=20$, $k=\{2-9\}$, and $r=\{5, 10, 20, 40\}$. Moreover, output noise of 0%, 1%, 5% and 10% was introduced over the problems to determine how robust was the classification process towards noise. 5 different problems of each k -DNF configuration were generated, and each problem was run with 5 different seeds. Also, all these runs were performed using fixed default class 0, to prevent the system from learning the inverse problem, over which calculating the success of the heuristic would not be straightforward.

The learning process was not performed during this stage of experiments, but only the parameter setting stage. In these experiments we want to quantify how many times the heuristic finds the k of the problem (or at least a larger one) which ensures that the system will learn the problem.

We also experiment changing the parameter $minAcc$ (the minimum accuracy demanded in a rule to become a representative) to determine how this parameter affects the search, and show how it can help tackling problems with noise more efficiently. In these experiments we tested parameter values $minAcc = \{1.0, 0.95, 0.9\}$. To determine significant differences among these three variants we used a Friedman test with its post-hoc Holm test.

The rest of the parameters in our approach are shown in Table 7.1 for clarity and replication purposes. However, according to our preliminary experiments, the parameters shown in this table can be considered constants and they can remain fixed. Only the minimum accuracy $minAcc$ and the number of representatives R have an important impact on the results, because they are directly related to the problem noise and the additional search effort, respectively. For simplicity in the first experiments we have set the R parameter to a value (10) that in preliminary work showed to be suitable for all tested scenarios. Afterwards, we experiment with different values of R to determine how this parameter affects the success rate and the effort of

Table 7.2: Probability of success in k -DNF problems with different amounts of output noise and minAcc values. The cells in green and red show the cases where the success rate increased or decreased with respect to the base case $\text{minAcc} = 1$, respectively. The cells in bold emphasise the cases where the success rate is below 100%.

Prob d=20		minAcc=1.0				minAcc=0.95				minAcc=0.9			
		Noise Level				Noise Level				Noise Level			
k	r	0%	1%	5%	10%	0%	1%	5%	10%	0%	1%	5%	10%
2	5	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	72.00	100.00	100.00	100.00
2	10	100.00	0.00	0.00	100.00	32.00	64.00	100.00	100.00	0.00	4.00	20.00	100.00
2	20	80.00	4.00	0.00	0.00	32.00	0.00	100.00	0.00	32.00	0.00	24.00	100.00
2	40		0.00	0.00	0.00		0.00	80.00	0.00	0.00	4.00	0.00	96.00
3	5	100.00	100.00	24.00	0.00	100.00	100.00	100.00	4.00	96.00	100.00	88.00	100.00
3	10	100.00	84.00	44.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
3	20	100.00	0.00	0.00	0.00	16.00	80.00	100.00	0.00	0.00	0.00	12.00	100.00
3	40	100.00	0.00	0.00	0.00	0.00	0.00	100.00	0.00	0.00	0.00	0.00	100.00
4	5	100.00	100.00	68.00	4.00	100.00	100.00	100.00	28.00	100.00	100.00	100.00	100.00
4	10	100.00	96.00	16.00	0.00	100.00	100.00	100.00	0.00	100.00	100.00	100.00	100.00
4	20	100.00	4.00	0.00	60.00	100.00	100.00	100.00	60.00	92.00	100.00	100.00	100.00
4	40	100.00	0.00	0.00	0.00	4.00	24.00	100.00	0.00	0.00	0.00	4.00	100.00
5	5	100.00	100.00	100.00	44.00	100.00	100.00	100.00	44.00	100.00	100.00	100.00	100.00
5	10	100.00	100.00	64.00	0.00	100.00	100.00	100.00	8.00	100.00	100.00	100.00	100.00
5	20	100.00	60.00	8.00	0.00	100.00	100.00	100.00	0.00	100.00	100.00	100.00	100.00
5	40	100.00	4.00	0.00	0.00	100.00	100.00	100.00	0.00	92.00	80.00	100.00	100.00
6	5	100.00	100.00	100.00	60.00	100.00	100.00	100.00	88.00	100.00	100.00	100.00	100.00
6	10	100.00	100.00	100.00	28.00	100.00	100.00	100.00	8.00	100.00	100.00	100.00	100.00
6	20	100.00	100.00	48.00	0.00	100.00	100.00	100.00	0.00	100.00	100.00	100.00	100.00
6	40	100.00	40.00	0.00	0.00	100.00	100.00	80.00	0.00	100.00	100.00	100.00	84.00
7	5	100.00	100.00	100.00	88.00	100.00	100.00	100.00	92.00	100.00	100.00	100.00	100.00
7	10	100.00	100.00	100.00	28.00	100.00	100.00	100.00	48.00	100.00	100.00	100.00	100.00
7	20	100.00	88.00	100.00	4.00	100.00	100.00	100.00	16.00	100.00	100.00	100.00	92.00
7	40	100.00	76.00	4.00	0.00	100.00	100.00	92.00	0.00	100.00	100.00	100.00	76.00
8	5	100.00	100.00	100.00	80.00	100.00	100.00	100.00	76.00	100.00	100.00	100.00	96.00
8	10	100.00	100.00	100.00	68.00	100.00	100.00	100.00	72.00	100.00	100.00	100.00	100.00
8	20	100.00	100.00	100.00	4.00	100.00	100.00	100.00	12.00	100.00	100.00	100.00	80.00
8	40	100.00	84.00	92.00	0.00	100.00	84.00	100.00	0.00	100.00	84.00	100.00	44.00
9	5	100.00	100.00	84.00	48.00	100.00	100.00	92.00	24.00	100.00	100.00	88.00	68.00
9	10	100.00	100.00	96.00	68.00	100.00	100.00	100.00	72.00	100.00	100.00	100.00	84.00
9	20	100.00	100.00	100.00	36.00	100.00	100.00	100.00	32.00	100.00	100.00	100.00	92.00
9	40	100.00	100.00	80.00	4.00	100.00	100.00	80.00	8.00	100.00	100.00	72.00	44.00

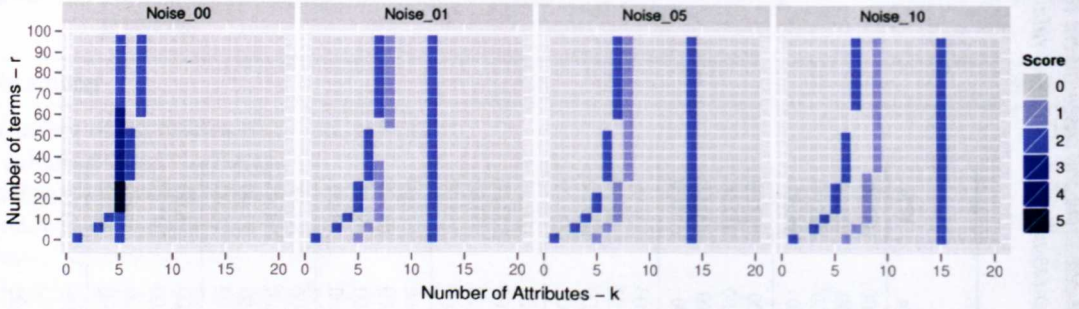


Figure 7.4: Final score grids in a problem with $d = 20$, $k = 5$ and $r = 20$ for the 4 levels of noise

the heuristic. Moreover, it is worth noticing that the reason why the $P(rep)$ function has a score lower than the two other metrics is because, in preliminary experimentation, this metric was not as reliable as the other two metrics.

7.4.1.1 Results

Table 7.2 presents the results for the different k-DNF configurations and different values of $minAcc$ in terms of percentage of success (finding the k of the problem or at least a larger one). The cells emphasised represent the configurations where the success rate is less than 100%. For $minAcc < 1$, the cells marked with red show the cases where the success rate is lower than the base case ($minAcc = 1.0$), and the cells marked with green show the cases where the success rate increased. In this table we can observe that using $minAcc = 1$ the heuristic is able to find the appropriate coverage breakpoint for most of the configurations with no noise. Moreover, it is noticeable that the output noise affects the performance of the heuristic.

On the other hand, we can observe that the heuristic fails in the cases where there is very high overlapping. These problems are very difficult to solve (as shown in Chapter 5) by the system because of the class imbalance, so it is not surprising that they are difficult for the heuristic as well. Such large overlapping makes the heuristic think that the k is smaller than the real one. In the case of a synthetic problem like the k-DNF where we know the correct answer, this is incorrect. However, what the system is trying to do is not completely wrong, because it is trying to solve the problem with less complex rules compromising the accuracy slightly, which in real-life domains can be advantageous. To understand better these domains further research focused specifically on rule overlapping is needed.

Figure 7.4 shows an example of the score grid for a problem with $k = 5$ and $r = 20$ with the 4 levels of noise and using $minAcc = 1$. The vertical stripe corresponds to the 2 points that are awarded to the most frequent number of attributes observed in the representatives. The other two other areas correspond to the scores awarded by the other two criteria. In this figure we can see that while the problem increases the representatives present a higher number of attributes. When this happens the area corresponding to the vertical stripe does not intersect the two other areas, thus the heuristic fails to find the appropriate k value. This is because the constraint of $minAcc = 1.0$ is too strict in these cases where the problem has noise. In the cases with noise we should take in consideration that good rules will have a good accuracy, but not equal to 1. Relaxing this parameter, as we can see in Table 7.2, helps finding adequate representatives for problems with noise. We can observe here that the success rate increases in most cases, except when the problem has no noise. As expected, when the problem has 5% noise, the best results are obtained using $minAcc = 0.95$, and the same occurs when the problem has 10% noise and we use $minAcc = 0.9$.

Moreover, Table 7.3 contains the statistical analysis to determine which value of $minAcc$ produces better results depending on the amounts of noise. In this table we can observe that the

Table 7.3: Results of the Friedman test performed to determine the best minAcc value depending on the noise. Control shows which was the algorithm that obtained the best ranking. Dominance shows the configurations that are significantly worst than the control configuration according to the Holm post-hoc test with $\alpha = 0.05$.

d	Noise	p-value	Control			Dominance
			1.0	0.95	0.9	
20	0%	0.00050	*			0.9
20	1%	0.00100		*		1.0
20	5%	1.895e-07		*		1.0, 0.9
20	10%	1.119e-11			*	1.0, 0.95

control method changes as expected depending on the noise of the problem, and for the problems with high amount of noise (5% and 10%) the respective control method performs better than the rest of the configurations.

Based on these results it is possible to state that for problems with noise we should use a minimum accuracy equal to the percentage of noise observed in the problem ($\text{minAcc} = 1 - \text{Noise}$). Even though this introduces a new parameter minAcc , the percentage of permitted noise is a much more intuitive parameter to set up than the coverage breakpoint, since it is an structural parameter of the problem, instead of being a parameter of the system.

One interesting thing to notice in Table 7.2 is that for problems with larger k and either too many or too few terms (even when using the adequate minAcc value), the heuristic seems to fail. When the problem has too many terms, it is possible to find representatives but these representatives are likely to have a large k and this value might not intersect with the other two areas, as it was exemplified in Figure 7.4. Moreover, if the problem has few terms finding a representative becomes very difficult and it is possible that the system does not find any representatives during the search process. In this case the mechanism will only rely on the number of negative examples to make a decision.

Our hypothesis is that these difficulties can be overcome by changing the selection policy of the k when there is not a single cell where the three metrics have intersected. Moreover, adjusting the minimum accuracy along the search process or a more granular step size in the adaptation of p could help obtain better results by finding representatives when this task is very difficult. More experimentation is needed to validate these hypotheses.

Based on these results, we can conclude that our parameter control mechanism is able to find the appropriate k value for a wide variety of binary problems, including problems with noise. However, other aspect in the parameter control we are yet to analyse is the overhead incurred by the local search mechanisms used (i.e representative search and pruning).

7.4.1.2 Computational effort of the heuristic

As we already explained in previous sections, our parameter control method, involves an additional computational effort before the learning process. This extra effort of the heuristic includes the evaluation of the randomly initialised individuals used to find representatives plus the number evaluations needed to prune the representatives.

Figure 7.5 shows the additional effort incurred by our approach for all the scenarios analysed in the previous section. The effort is shown in terms of number of rule evaluations (matching the rule with the complete training set and computing its accuracy). Execution time is not shown as it is proportional to the number of evaluations. In this figure we can observe that in

general while the k increases it becomes more expensive to run the parameter setting approach, as more iterations are needed to find representatives. We can also observe that for high levels of noise and when using the wrong $minAcc$ the number of iterations to find representatives also increase. Also we can observe that in most cases while the r of the problem increases less iterations are necessary since it is easier to find representatives.

Moreover, it is also noticeable a spiking behaviour in the additional effort. This behaviour is clarified by Figure 7.6, which shows the frequency in which each value of p is selected. As it was explained before, our approach also adapts the p value to increase the probability of finding representatives when this task becomes very difficult. As we can see, the different stages in the behaviour of the effort observed in Figure 7.5 correspond to the transition stages between different values of p . When the system uses a smaller p the additional effort to find representatives becomes smaller. This is because, as shown in Chapter 6, as k increases the adequate p to solve the problem decreases. We can also observe that in the most difficult cases which are $Noise > 1 - minAcc$ the system tends to reduce the value of p more than in the cases where the parameter $minAcc$ is setup correctly.

Furthermore, the effort incurred by our approach can in theory be reduced or increased by changing R , the number of representatives needed to stop the search. To understand better the effect of this parameter we performed additional experiments with a subset of the k-DNF problems analysed before ($k=5$, $r=20$). We tested different numbers of representatives $R=\{1,3,7,10,15,20\}$ and $minAcc=\{1.0,0.9\}$ (to show the impact over problems with noise). Our results are shown in terms of success ratio and additional number of evaluations performed. Moreover, we show the Pareto frontier between the effort and the success rate, to determine the trade off between the extra evaluations and quality of the solution.

Figure 7.7a shows the success rate with respect to the number of representatives. It is worth mentioning that in the previous experiments, the configurations where $Noise > 1 - minAcc$ were not solved. It is noticeable in this figure that when the $minAcc$ parameter is configured correctly according to the noise the number of representatives needed to find the right coverage breakpoint value is very small. On the other hand, Figure 7.7b presents the number of extra evaluations performed when using different number of representatives. Here it is possible to observe that the number of evaluations increase linearly with the number of representatives, except for the case with $n=10\%$ and $minAcc=1$. Also the evaluations increase depending on the amount of noise, since this makes the problem more difficult. More interestingly, looking at the Pareto frontier in Figure 7.7c, it is noticeable that ≈ 10000 additional evaluations are needed to ensure success for this problem. Considering the number of evaluations reported in the previous section, we can observe that by reducing the amount of representatives needed it is possible to reduce the additional effort by more than half in the worst case.

7.4.2 Analysis over a real-world PSP problems

In order to test the performance of this method over real-world domains, we selected a binary protein structure prediction problem *CN-bin*. It is worth mentioning that this problem has a very high noise ratio, and there are no possible rules that have 100% accuracy. Therefore, in order to test the heuristic, the minimum accuracy required for the classifiers to be representatives was relaxed.

In this section we are going to analyse the results obtained with our approach by comparing them with an exhaustive experimentation to determine the adequate coverage breakpoint value. In the exhaustive experimentation to hand-tune the coverage breakpoint, we used values ranging from $1/2^2$ to $1/2^9$ (as this is the maximum possible since the problem has 9 attributes), and different values of p (0.75, 0.5 and 0.25). Since we are dealing with a real problem in this section, the system is going to perform the whole learning process using the coverage breakpoint selected and the success will be determined based on the final accuracy. The parameters

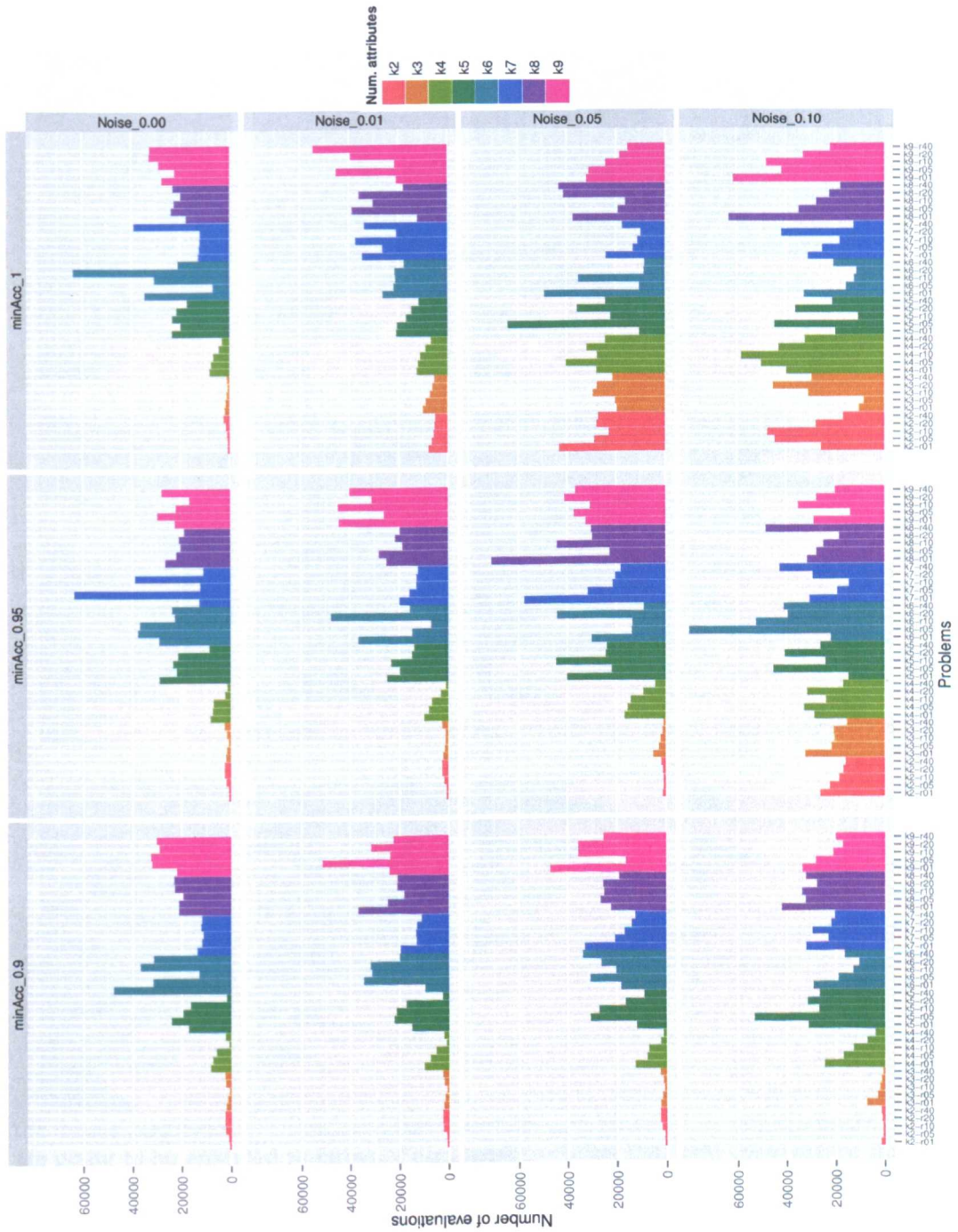


Figure 7.5: Number of additional evaluations performed to determine the coverage breakpoint value in each scenario

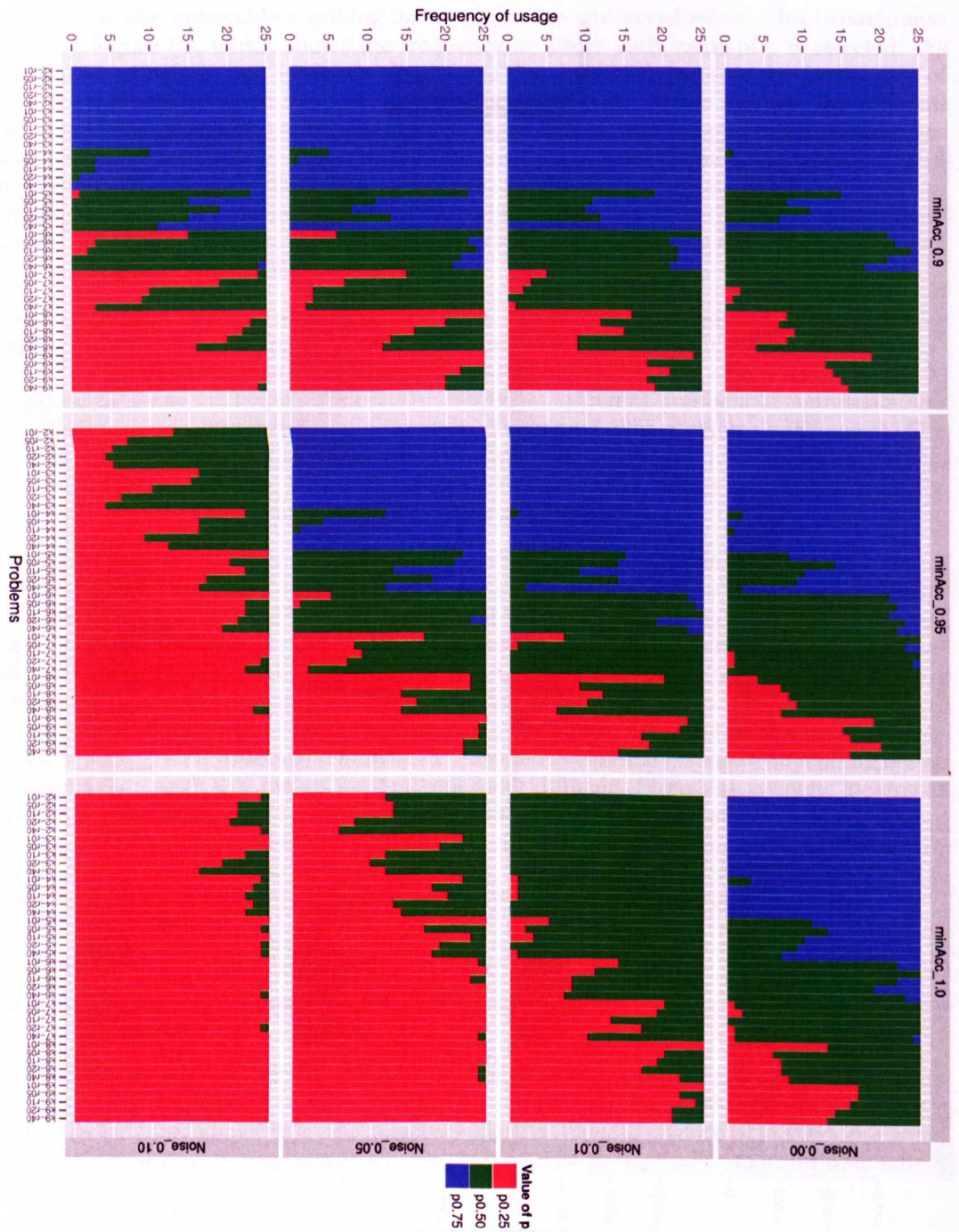


Figure 7.6: Frequency of usage of the different p values to determine the coverage breakpoint value

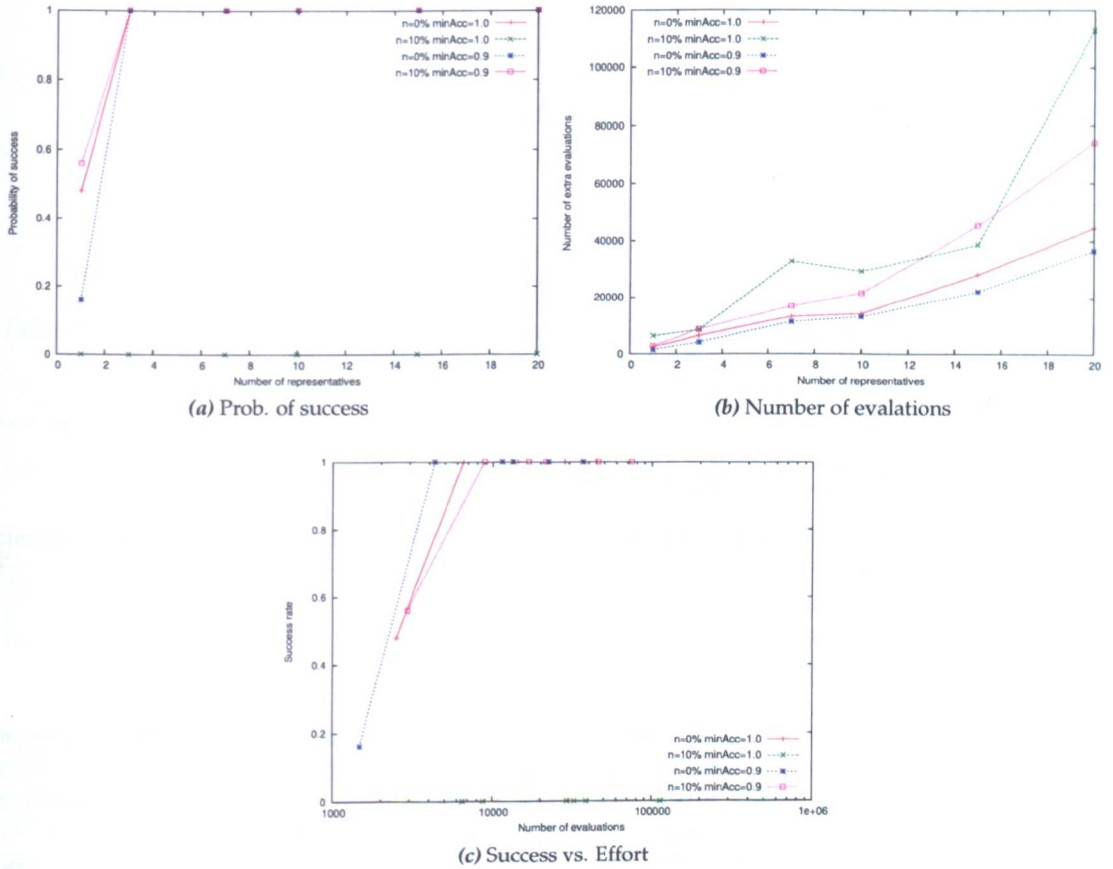


Figure 7.7: Probability of success and number of additional evaluations using different number of representatives R over k -DNF problems with $d=20$, $k=5$, $r=20$ and $\text{minAcc}=1.0$.

Table 7.4: Parameters for the BioHEL system different than its fixed configuration.

BioHEL Parameters	Value	Heuristic Parameters	Value
Default class	fixed 0	Maximum number of rules considered	1000
Number of windows in ILAS	20	Rule step size	20

for the BioHEL system are the ones presented in Section 3.3, except only for three parameters shown in Table 7.4. Moreover, this table also shows the parameters of the heuristic that vary from the ones presented in the previous section.

For the analysis of the results, we will first apply a Wilcoxon pairwise test to determine if there are significant differences between using different coverage breakpoints in this problem and which is the most adequate parameter value to solve it. Having found the adequate coverage breakpoint value, we will then compare it with the results found by our approach. The results are going to be analysed in terms of accuracy, execution time and convergence time of both methodologies.

Regarding the dataset, this problem was separated in 10 training and test sets for ten-fold cross-validation and each one of the training sets contains approximately 233812 instances. Moreover, each one of these problem instances was run with 5 different seeds, so the results are the average of 50 runs.

Table 7.5 shows the results in terms of accuracy, average time per run and total time of the experiments when applying different coverage breakpoints of the type $\text{cov} = 1/2^k$ where $k = \{2..9\}$.

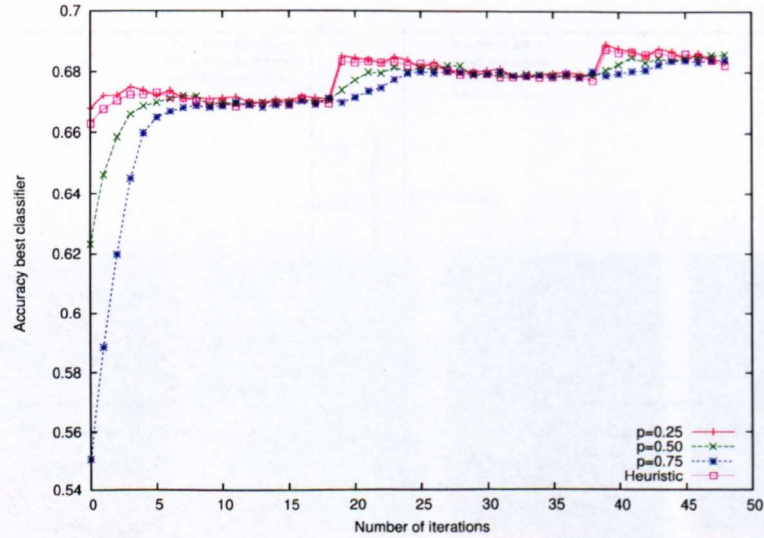


Figure 7.8: Average accuracy of the best classifier during the 50 iterations of the GA using different values of p .

In this table, we can see that the coverage breakpoint that obtains the best results in terms of accuracy is $1/2^9$. Also, for this problem there are no differences between using different values of p in terms of accuracy or execution time. Moreover, in Table 7.6 we can see that the coverage breakpoint 2^{-9} does not produce results significantly different than the results applying other values such as 2^{-7} or 2^{-8} . However, the maximum average accuracy is obtained with the smaller coverage breakpoint.

In Table 7.7 we can observe the results obtained with our approach. Using a minimum accuracy of 0.7 the system determined that the k of the problem is equal 9 in the majority of the cases and applying the corresponding coverage breakpoint we obtain an accuracy equal to our best results in preliminary experimentation. We also tested a more relaxed *minAcc* value without obtaining good results.

*Based on these results we can conclude that the heuristic, when using the appropriate noise ratio, is able to categorise this real problem correctly and determine the appropriate coverage breakpoint automatically. Regarding the computational time, the total amount of CPU time invested in performing the preliminary experiments with the different coverage breakpoints and different values of p is equal to ≈ 285 hours. On the other hand, the amount of time invested in running the experiments setting the coverage breakpoint automatically, with different *minAcc* values and including the whole learning process was ≈ 81 hours. This constitutes 28% of the time invested in the preliminary experiments, which means a reduction of 71% in the total experimental time.*

Moreover, our method also adapts the value p , the probability of setting the bits in the GABIL representation to 1. Figure 7.8 reports the average accuracy of the best rule along the 50 iterations of the GA with different p values. So far the use of adapting this parameter was just to find representatives during the parameter control stage, but the adaptation can also help finding good rules faster within the GA. As Figure 7.8 shows, using $p = 0.25$ makes the system find good classifiers quicker (although all three values of p eventually manage to learn the correct rules). This is because in this particular problem using a small value of p increases the odds of finding representatives. The spiking behaviour in the figure is a normal phenomena due to the usage of ILAS windowing scheme [Bacardit, 2004], and it is not related to the approach presented in this chapter.

Table 7.5: Results over the binary CN-bin problem using fixed coverage breakpoints and different values of p . The table shows for each p and coverage breakpoint value the average accuracy obtained, the average experimental time of each run and the total experimental time.

	Cov (k)	Accuracy (%)	Average time (s)	Total time (s)
$p = 0.75$	2	70.15 ± 0.96	244.67 ± 120.89	12233.71
	3	70.27 ± 0.84	427.00 ± 207.95	21350.19
	4	71.46 ± 0.56	378.64 ± 207.36	18932.08
	5	71.79 ± 0.47	386.40 ± 147.90	19319.86
	6	72.12 ± 0.45	640.26 ± 238.88	32013.22
	7	72.32 ± 0.52	874.15 ± 322.80	43707.67
	8	72.40 ± 0.47	1469.14 ± 496.96	73457.19
	9	72.45 ± 0.47	2526.63 ± 723.67	126331.39
	Total time			(≈ 96.48 h) 347345.31
$p = 0.5$	2	70.15 ± 0.96	247.72 ± 101.81	12385.91
	3	70.27 ± 0.84	342.26 ± 180.62	17112.77
	4	71.47 ± 0.56	390.89 ± 208.98	19544.62
	5	71.79 ± 0.47	398.20 ± 180.34	19909.83
	6	72.12 ± 0.45	639.17 ± 240.29	31958.73
	7	72.31 ± 0.53	896.59 ± 287.78	44829.29
	8	72.40 ± 0.48	1601.17 ± 445.91	80058.71
	9	72.45 ± 0.47	2244.94 ± 650.95	112246.99
	Total time			(≈ 93.90 h) 338046.85
$p = 0.25$	2	70.46 ± 0.84	239.53 ± 91.12	11976.29
	3	70.27 ± 0.84	362.16 ± 191.81	18107.84
	4	71.46 ± 0.56	332.23 ± 183.90	16611.51
	5	71.80 ± 0.48	409.98 ± 167.93	20498.96
	6	72.12 ± 0.45	622.01 ± 230.56	31100.48
	7	72.31 ± 0.53	846.66 ± 272.08	42333.25
	8	72.40 ± 0.48	1636.27 ± 482.24	81813.37
	9	72.45 ± 0.47	2338.56 ± 586.86	116927.77
	Total time			(≈ 94.26 h) 339369.47

Table 7.6: *P-values of the Wilcoxon pairwise test to determine significant differences between the usage of different coverage breakpoints of the type 2^{-k} in the binary CN-bin problem. The cells in bold indicate the cases where significant differences were found.*

P-values of the wilcoxon pairwise test							
k	2	3	4	5	6	7	8
3	0.52092	-	-	-	-	-	-
4	2.9e-07	1.2e-08	-	-	-	-	-
5	1.5e-11	2.9e-11	0.01510	-	-	-	-
6	4.9e-15	4.9e-15	2.5e-06	0.00265	-	-	-
7	< 2e-16	< 2e-16	1.0e-08	0.00016	0.21071	-	-
8	< 2e-16	< 2e-16	6.1e-10	6.1e-06	0.05857	0.57281	-
9	< 2e-16	< 2e-16	7.5e-11	2.8e-06	0.01079	0.21071	0.57281

Table 7.7: *Performance of the heuristic over the binary CN-bin problem. The columns show the minimum accuracy used, the average k and p obtained by the heuristic, the test accuracy after learning with the corresponding coverage breakpoint value and the total experimental time.*

Min Acc	k	p	Test acc	Ave. Exec Time	Total time
0.7	8.98±0.14	0.28±0.08	72.45±0.47	2913.93±1356.41	145696.66
0.6	5.62±1.12	0.73±0.06	71.94±0.74	902.29±382.38	45114.63
Total time				(≈80.94 h)	291393.32

In this sense, we can say that the heuristic reduces the computational time needed to set up the algorithm properly, which for large problems can constitute a considerable amount of CPU time. Moreover, it adapts other parameters of the system, such as the p value, which helps finding good rules quicker within the GA.

7.5 Conclusions and Further Work

In this chapter we have presented an automatic procedure to learn the structure of the problem and set the coverage breakpoint parameter in the BioHEL learning system for (a broad range of) binary classification domains. Our procedure is able to estimate the problem's structure (k and r) in binary problems with and without noise and in real-world problems such as protein structure prediction problems. Our approach exploits the theory developed in Chapters 5 and 6 over the BioHEL system within the binary domain to classify problems according to their the class imbalance, probability of generating representatives in randomly initialised sample of individuals and other characteristics observed in the sampled individuals along the search process. Using this method we were able to set up the coverage breakpoint parameter in BioHEL, facilitating in this way the learning process, reducing the computational time of preliminary experiments and making the system easier to use for an end-user. The final validation stage using a challenging protein structure prediction problem showed that the heuristics works for real world problems as well, and that it is possible to reduce the total experimental time by 71%. Moreover, our approach also adapts other parameters like p , the probability of settings bits to 1 in the GABIL representation, which can help finding good rules faster within the GA.

Even though the model for the probability of obtaining representatives used is only applicable to the GABIL representation within ALKR, we think that the methodologies presented in this chapter can be extended to other systems by using the models suitable for their knowledge representations. Also, this methodology is not only useful to determine parameters within BioHEL's fitness function, but also knowing in advance the characteristic of the problem at the beginning of the learning process can be advantageous to guide the search and also set parameters within the system.

As a further work we would like to extend this work to χ -ary discrete and continuous domains. However, in these cases further post-processing mechanisms might be necessary to refine the good rules since the relationship between coverage and number of attributes is not strictly linear. In the case of the χ -ary domain, the coverage would not only depend on k but would also depend on e the number of values activated in an attribute (number of 1s in the GABIL representation), while in the real domain the coverage will depend on the size of the intervals.

Moreover, further research will be focused in develop different policies for selecting the k when there is not a single cell where the three metrics have intersected. In these cases probably it is better to select a k based on the average of the cells that obtained the highest score.

Finally, it would be interesting to recycle the representatives found within the learning process by forcing these classifiers to appear in the initial populations of the subsequent GAs. Including these good rules in the following initial populations can improve the convergence time of the GA, by assuring a percentage of good individuals to work with during the search process.

Fast evaluation process using GPGPUs

This chapter is focused on reducing the execution time of BioHEL, which tends to be high (as in most supervised learning approaches) when handling large scale datasets. In order to do this a method for computing the fitness in BioHEL using NVIDIA CUDA is introduced. Both the match process of a population of classifiers against a training set and the computation of the fitness of each classifier from its matches have been parallelised. This chapter discusses the results obtained using two different parallelisation paradigms: a coarse-grained parallelism (which parallels the evaluation of each classifier independently) and a fine-grained one (which performs the match of each attribute in the classifier in parallel). Moreover, this chapter discusses how the speedup increases when this approach is combined with other efficiency enhancement mechanisms, how the execution times obtained can be explained by simple models and how this approach can be easily extended to other EL systems.

8.1 Introduction

As we observed in Chapter 4 one of the weaknesses of the BioHEL system is its execution time. This is due to the fact that, as in most EL systems, the match process of the whole population in BioHEL is computationally very expensive, specially when handling large scale datasets. Moreover, as the problems become more complex the execution time increases since the system needs to learn more rules. The objective of this chapter is to speed up the evaluation process¹ in BioHEL to overcome these problems.

There is extensive efficiency enhancement work in the evolutionary computation context as it was already mentioned in the background material of this thesis. Moreover, in the last years, the usage of General-purpose Graphics Processing Units (GPGPUs) has become a popular practice in high performance computing. By exploiting hardware originally designed to render 3D graphics at high speed it is possible to perform highly parallel general purpose computations. GPGPUs have been used already to speed up the evaluation process in GAs [Maitre et al., 2009; Pospichal et al., 2010], genetic programming [Chitty, 2007; Harding and Banzhaf, 2007; Langdon and Harrison, 2008] and LCS [Lanzi and Loiacono, 2010; Loiacono, 2011], as it was mentioned in Section 2.3.5.3.

In this chapter an efficient GPGPU-based parallel fitness computation process for the BioHEL system is presented. Specifically, for its implementation NVIDIA CUDA [NVIDIA, 2008] is used. The goal of this approach is to calculate the accuracy and coverage metrics (which are the base of BioHEL's fitness function) as fast as possible. Two different implementations are analysed. The first one, which will be referred as *coarse-grained parallelisation*, calculates the

¹Since BioHEL uses a supervised learning approach, we will refer to the match process as the evaluation process for the rest of the chapter.

match between all the rules in the population and all the examples in the training set in parallel. The second one, which will be called *fine-grained parallelisation*, expands the parallelism of the previous one to a third dimension by performing the match of each attribute in the rules in parallel as well. After the match is computed, both implementations compute in parallel the accuracy and coverage metrics of the whole population by means of a parallel reduction algorithm.

We test our parallel implementations against BioHEL's serial version using thirteen different problems with a broad set of different characteristics. First we analyse the speedup of the evaluation process independently and then we analyse the integration of the parallel fitness function within the whole learning algorithm.

We also evaluated the combination between our CUDA implementations and the ILAS windowing scheme, since it is one of the main techniques used to achieve speedups in BioHEL (See Section 2.4.1.3). This comparison is aimed to determine whether the speedup obtained by both methods is complementary and thus can be successfully combined, or whether there is a trade off that might constraint their integration. Moreover, we generated empirical models that explain the execution times obtained with our CUDA-based approach depending on the training set size and the number of attributes of the problem.

This chapter is organised as follows. Section 8.2 describes the CUDA architecture and references previous works that use GPGPUs in ML. Section 8.3 describes the implementation of the CUDA-based evaluation process. Section 8.4 explains the experimental design followed. Section 8.5 shows the results of the evaluation process independently and Section 8.6 shows the results of the evaluation process integrated with BioHEL. Moreover, Section 8.7 shows the execution time models for our CUDA approach. Finally, Section 8.8 presents the final remarks and further work.

8.2 GPGPUs and CUDA

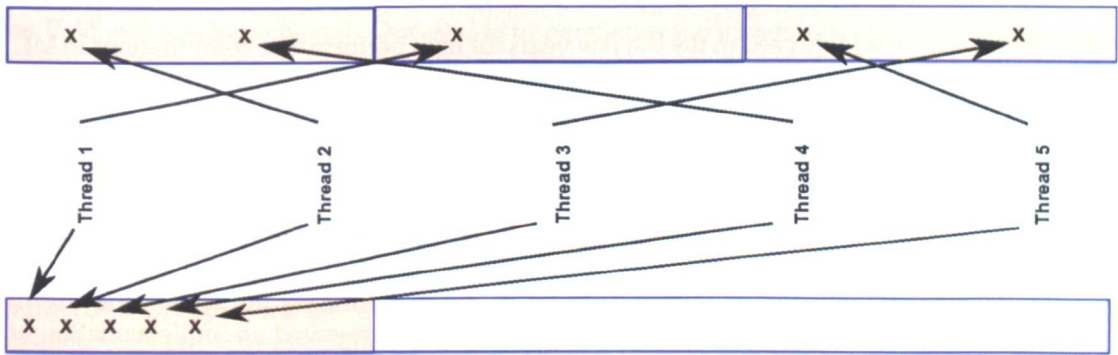
The programming tool used to parallelise the evaluation process in BioHEL is the Computer Unified Device Architecture (CUDA) [NVIDIA, 2008], a parallel computing architecture developed by NVIDIA. CUDA allows the user to exploit in a general purpose manner the computing capacity inside the NVIDIA GPGPUs. In this chapter we will refer the GPGPU as *device* and the computer in which it is installed as *host*.

CUDA follows the Single Instruction Multiple Data (SIMD) paradigm which consists in running the same operations in each one of *threads* over different data. The thread is the most granular processing unit inside a GPGPU. Each one of the threads run within a *block*, which is the minimum unit processed by a single multiprocessor inside the GPGPU. All the threads inside the same block are able to share information through the *shared memory*. On the other hand sharing memory among blocks is only possible through global synchronisation. Moreover, the number of blocks that run in parallel in the GPGPU depends on the specifications of the card and the amount of shared memory the blocks need to run.

CUDA provides an extension of the C language to implement the *kernel functions*, which is the code that will run inside the threads. The kernel functions work in the device memory, reason why it is necessary to copy the data we need to process from host to device and then copy the results from device to host. Allocating memory in the device and copying in and out of it can be computationally very expensive and it is necessary to take this limitation in consideration.

Moreover, the wise usage of the memory in CUDA is crucial for the performance of the algorithms. There are five types of memory in CUDA, each of them with a different aim. If any of these types is not properly used, the performance drops dramatically. Furthermore, *coalescence*

Non-coalesced access (3 read operations)



Coalesced access (1 read operation)

Figure 8.1: Example of coalesced and non-coalesced memory access.

is also important when accessing the memory in CUDA. Coalescence is achieved when the consecutive threads in a block are accessing consecutive positions in memory. Since CUDA reads memory in chunks having a not coalesced memory access makes the system read unnecessary parts of the memory as shown in Figure 8.1. This increases the execution time of the kernels, since more read operations will be needed to read the same amount of data [NVIDIA, 2008; Sanders and Kandrot, 2010].

The different types of memory in CUDA, ordered by access speed from slowest to fastest, are:

- **Device memory:** is the part of the memory where all the data structures manipulated by the kernels are stored. It is not cached, so this memory is readable and writable from the device and from the host, and it depends on the total amount of memory the video card has. From the point of view of the threads this memory is also known as *global memory*, because it is readable from threads in different blocks. However, it cannot be used to send information between threads in different blocks since there is no guarantee that one block will be executed before another.
- **Texture memory:** is not a different part of the memory, but a different way of accessing the global memory. Memory that is declared as textures is accessed through a read-only cache which is optimised for spatial locality instead of memory locality. This part of the memory is usually used to render textures and surfaces within graphic cards, and it is not used within our evaluation approach.
- **Shared memory:** is the memory shared by all threads in one block. This memory is faster than the previous one and it is commonly used to share information between threads of the same block and perform preliminary computations at this level. If the access to this memory is coalesced then the access speed will be as fast as accessing registers.
- **Constant memory:** is a part of the device memory visible by all the blocks but not writable by the threads. This part of the memory can only be written by the host. This type of memory should only be used to store information that is accessed constantly and does not change during the execution of the program. Its usage is very limited due to the small amount of constant memory in the GPGPUs.
- **Local thread memory:** is the part of the memory only visible by the thread itself, also known as registers. The variables created inside the kernel code are stored in this memory. There is limit in the number of registers a thread can use without degrading the performance. If the kernel uses a large number of registers then the data is stored in global memory, which is slower to access.

8.2.1 GPGPUs in Machine Learning

GPGPUs have been widely used in the last few years for high performance computations in ML. For instance, they have been already used to speedup self-organising maps for pattern classification [Prabhu, 2008], decision trees [Sharp, 2008], neural networks [Steinkraus et al., 2005], and support vector classification [Catanzaro et al., 2008]. GP and GAs have also benefited from the availability of GPGPU hardware. Chitty [2007] and Harding and Banzhaf [2007] independently presented the first applications of GPGPUs to GP, which were developed contemporarily. These works were a proof of concept and they highlight how the parallelisation should be done. However, they do not use large datasets for testing purposes. Afterwards, Langdon and Harrison [2008] implemented the evaluation process of GP trees for bioinformatic purposes achieving an speedup of approximately 8X. Moreover, Maitre et al. [2009] presented an implementation of a GA which performs in parallel the evaluation function of all the individuals using CUDA. The major difference between this work and our approach is that they do not have a training set but a function instead, which they run in parallel over the different individuals. Our fitness computation is based on managing a training set within the device (which implies a higher memory occupancy) while they use a compact mathematical (algorithmic) representation as the fitness function.

Also, [Pospichal et al., 2010] presented a GA that runs in CUDA following the island parallelisation model (See Section 2.3.5.2). To implement this paradigm in CUDA each one of the blocks correspond to an island and each one of the threads correspond to an individual. Moreover, to apply global operators it is necessary to perform a global synchronisation. Also the same as [Maitre et al., 2009], this approach does not use a training set but a mathematical function to evaluate the individuals.

Moreover, [Li et al., 2007; Yu et al., 2005] present fine-grained parallelisation approaches implemented in GPGPUs. In these approaches the whole GA is implemented within the device. This means that each thread holds the information of a single individual and the communications are restricted to high speed communications (neighbour threads).

Regarding speeding up LCS, Lanzi and Loiacono [2010] presented a work which parallelises the creation of the prediction array in XCS. This work was mainly focus on improving the match of interval attributes by means of bitwise operators following the methodologies in [Butz et al., 2008a; Dorigo and Colombetti, 1997]. This work produced from 3 to 12X speedup in the interval-based representations and between 20 and 50X speedup in the ternary representation. Later, this work was extended to apply the concepts presented in this chapter [Loiacono, 2011], which is to apply a reduction step to reduce the amount of the output structures and perform more calculations inside the GPGPU. In this study the speedup of interval-based representations was higher than before and it ranged between 2X and 32X.

8.3 CUDA-based efficient fitness computation process

As it was mentioned in the introduction, the fitness calculation is one of the major bottlenecks in terms of execution time for BioHEL, as well as for most EL systems. Computing the accuracy and recall of each rule is the most computationally demanding part of BioHEL's fitness function. To calculate these two values we need to compute three metrics per classifier:

1. The number of instances in the training set that match the condition of the rule.
2. The number of instances in the training set that match the class of the rule.
3. The number of instances in the training set that match the class and the condition at the same time.

The evaluation process of this system has a huge computational cost, because it involves comparing all the rules in the population with all the examples in the training set or strata (if using the ILAS windowing scheme). Our goal is to parallelise the match of all the rules in the population with each one of the examples and to calculate the results for each rule in parallel. Implementing this parallelism involves a greater challenge than the one presented by Maitre et al. [2009], because it involves not only individuals (rules in this case) but a training set as well. However, our learning paradigm has potentially a higher degree of parallelism, because we can perform the match with each instance of the training set in parallel.

Considering that the population size is n and the training set size is m , we intend to make $n \times m$ operations in parallel. The most naive way to do this is to perform the match operations in the GPGPU, copy back the results of each match to the host and then count the matches and mismatches there. However, this approach would be very slow because every time we calculate the fitness it will be necessary to copy a structure of size $O(n \times m)$ from device memory to host memory. This is very undesirable because the execution time of the memory copy operations is proportional to the structure size. Therefore, the most feasible way to avoid this computational cost is to calculate the final result for each classifier inside the GPGPU (and in parallel). By doing this, we will only need to copy a structure of size $O(n)$ containing the final results.

At this point our calculation involves two steps, which correspond to the main tasks of each kernel:

1. Calculating the match between all the classifiers and all the instances
2. Calculate the final results per classifier by means of a parallel reduction (adding all the elements within an array in parallel)

But as we explained previously the kernels work only on data stored in the device memory, so it is necessary to flatten the data, copy the data into the device and get the results back to host as well. Moreover, it is necessary to calculate if all the data is going to fit in the device memory. Considering this, our fitness calculation involves five stages as shown in Figure 8.2. However, the evaluation process step changes depending on the parallelisation paradigm used, since two different paradigms were implemented. The first one, *coarse-grained parallelisation* compares all the rules and the instances in the training set in parallel. The second one, *fine-grained parallelisation*, adds another dimension of parallelism to the previous approach by parallelism the match of the rules by its attributes.

Furthermore, the data in the training set and in the population does not stay constant during the execution of the algorithm. This is because every time we learn a rule some examples are deleted from the training set and also, during the learning process the population changes through crossover and mutation. This makes the implementation even more complex because data will need to be copied multiple times into device memory and these operations are very slow. The implementations presented in this chapter address this problem by attempting to minimise the copies into device memory. Considering that in each GA run (learning a single rule) the instances remain constant, and the algorithm will only use a subset of the instances in each iteration, our implementation tries to store in device memory all the instances at the beginning of each GA run. Then the windowing is handled inside the device passing the *window offset* as a parameter of the fitness function. The window offset is the pointer on device memory in which the instances of that particular window are stored. If the global memory is not big enough to store all the instances then further memory calculations are performed as it will be shown later in this section. On the other hand, the classifiers need to be copied into device memory in each iteration. However, this does not involve a considerable computational cost because in most cases the populations occupy relatively small amounts of memory in comparison with the instances.

The following subsections explain first the evaluation process, in which we explain the two parallel approaches and the implementation details particular to each approach. We will start explaining the coarse-grained parallelisation and then we will explain the second approach

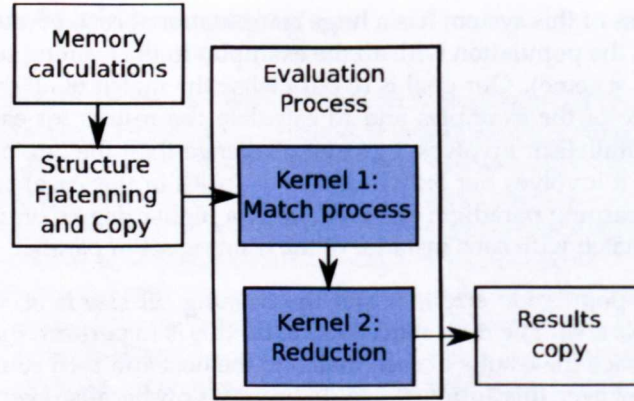


Figure 8.2: Stages of the CUDA evaluation in each iteration of the GA

focusing on the differences with the first one. Afterwards, we mention common strategies followed in the other stages.

8.3.1 Coarse-grained parallel approach

To perform the evaluation process in this approach we implemented two kernel functions. The first kernel is in charge of performing the match operations between the rules and the examples. Each thread carries out a single match operation and computes three boolean values per rule-example pair: a) condition match, b) class match and c) both condition and class match. The second kernel is in charge of counting the number of matches and mismatches of a classifier. That is, counting (in a general sense, reducing) the results calculated in the previous kernel. To do this we need to apply a reduction algorithm as it will be explained in Section 8.3.1.2.

However, since the threads in different blocks do not share memory the reduction is limited to the number of elements accessible in a block. If the problem is big it will take more than one step of reduction and global synchronisation to reduce the match information of a particular classifier with all the instances. So at the most if we reduce M items per block, the smallest number of items left to reduce would be y/M .

It would be intuitive to perform all the necessary reductions iteratively in the second kernel. However, this means writing to global memory a large amount of data (all the matches and mismatches per classifier) at the end of the first kernel, which has a large impact in the runtime. To minimise the volume of the output data the first stage of the reduction is already done in the first kernel in shared memory, before writing the values to the output memory. This means that if we reduce the results of 512 instances in each block the maximum output structure size that we need on the first kernel is:

$$x \lceil \frac{y}{512} \rceil A \quad (8.3.1)$$

where x is the number of classifiers, y is the number of instances and A corresponds to the output structure (12 bytes since we are storing 3 ints per pair). For this kernel the grid structure has 512 threads, which are used to check 1 classifier against 512 instances. This guarantees the minimum output structure possible at the end of this kernel.

This reduction performed in the first kernel is slower because it reduces three values at the same time (the condition match, the action match and the *and* between the previous two). Afterwards, the subsequent reductions (for each of the three metrics) are performed independently in the second kernel in a more efficient way. At the end of the execution of the second kernel we will have three values per classifier to copy back to host memory. Also, at the end of each

kernel the information is reorganised in order to minimise the run-time of subsequent kernels and the memory copy operations from device to host. The following subsections will explain more deeply the kernel functions of this approach and other implementations details, such as handling different types of attributes at the same time.

Algorithm 8.3.1: COMPUTEMATCH(*classifiers*, *instances*, *numIns*, *numClass*, *output*, *offset*)

Determine the thread index T_{ij} , the instance index I_j and the classifier index C_i to access the global memory

Declare *cond*, *action* and *match* arrays in shared memory

if $I_i < \text{numIns} \wedge C_i < \text{numClass}$

then $\begin{cases} \text{cond}[T_{ij}] \leftarrow 1 \\ \text{for } \text{att} \in \text{class}[C_i].\text{atts} \wedge \text{cond}[T_{ij}] \\ \quad \text{do } \text{cond}[T_{ij}] \leftarrow \text{match}(\text{ins}[I_j].\text{atts}[\text{att}], \text{class}[C_i].\text{atts}[\text{att}]) \\ \text{action}[T_{ij}] \leftarrow \text{ins}[I_j].\text{action} = \text{class}[C_i].\text{action} ? 1 : 0 \\ \text{match}[T_{ij}] \leftarrow \text{action}[T_{ij}] \wedge \text{cond}[T_{ij}] \end{cases}$

else if $\text{match}[T_{ij}] \leftarrow \text{action}[T_{ij}] \leftarrow \text{cond}[T_{ij}] \leftarrow 0$

Perform reduction over *match*[T_{ij}], *action*[T_{ij}], *cond*[T_{ij}]

comment: The first thread of a block copies the final value in global memory

if $j = 0$

then $\begin{cases} \text{Write } \text{match}[T_{ij}], \text{action}[T_{ij}] \text{ and } \text{cond}[T_{ij}] \text{ into global memory} \\ \text{output}[C_i][I_j] = \text{match}[T_{ij}] \\ (\text{output} + \text{offset})[C_i][I_j] = \text{action}[T_{ij}] \\ (\text{output} + 2 * \text{offset})[C_i][I_j] = \text{cond}[T_{ij}] \end{cases}$

8.3.1.1 Kernel 1: Computing the matches

The first kernel is in charge of performing, in parallel, the match operations between all the rules and all the examples. Algorithm 8.3.1 shows the pseudo-code for this function. After calculating the three match values in each thread they are stored in a shared memory structure over which a one-level parallel reduction will be performed. This reduction algorithm is based on the parallel reduction algorithm proposed by Harris [2007] (which will be explained in the next section) but, instead of reducing only one value, it reduces three values at the same time. Now instead of having an output structure of size $O(n \times m)$ we have one of size $O(b \times n)$ where b is the number of blocks used in this first kernel. This reduces the amount of time spent in writing in global memory as well as the amount of global memory needed.

	B1	B2	B3	B1	B2	B3	B1	B2	B3
C1	C	C	C	A	A	A	M	M	M
C2	C	C	C	A	A	A	M	M	M
C3	C	C	C	A	A	A	M	M	M
C4	C	C	C	A	A	A	M	M	M
C5	C	C	C	A	A	A	M	M	M

Figure 8.3: Distribution of the global memory by the end of the first kernel function. C = condition counter, A = class counter, M = total match counter. C_i corresponds to the classifier index and B_j corresponds to the block index.

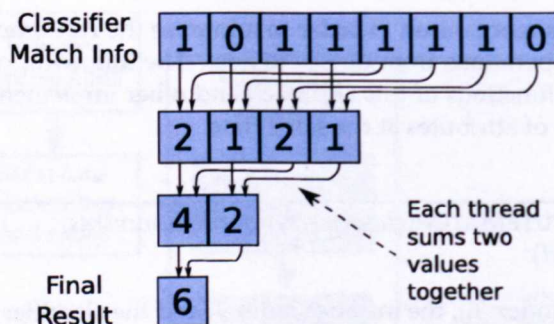


Figure 8.4: CUDA parallel reduction algorithm.

At the end of the execution of this kernel, the result values are copied back into three separate blocks as shown in Figure 8.3. This allows having three separate areas over which we can perform a more efficient reduction with the second kernel.

8.3.1.2 Kernel 2: Adding up the results in parallel

The second kernel is in charge of reducing iteratively the information that was previously calculated in the first kernel. This kernel will perform the CUDA parallel reduction algorithm [Harris, 2007] without any major modification. Figure 8.4 shows an example of the reduction process for the match information of one classifier. In this type of reduction each thread is in charge of adding up two values. To guarantee coalesced access when summing 2^k values the first 2^{k-1} threads with index t will add up their value with the corresponding thread $t + 2^{k-1}$. Then the second half of the threads become inactive and the first half of threads add their value with the corresponding thread index $t + 2^{k-2}$. This is done iteratively until only one value is left. When the number of elements that we need to reduce is not a power of 2, we simply fill the memory with data that will not affect the calculations. This kernel always launches 512 threads, 1 in the axis of the classifiers and 512 in the axis of the integers to reduce.

This reduction is applied independently over each one of the three memory areas created by the previous kernel. Preliminary experimentation showed that performing reduction of only one value is faster than performing reduction over three values at the same time.

This kernel is called iteratively until the number of blocks used is equal to one. In the last iteration the kernel will reorder again the data in memory copying all the information of each classifier next to each other. After that, the information of the classifiers will be pack together to make possible copying it back to host memory using a single memory copy operation. Algorithm 8.3.2 shows the pseudocode for this kernel and Figure 8.5 shows the complete flow of data in this approach.

8.3.1.3 Working with mixed attribute types

In the ALKR representation discrete and continuous attributes may coexist together. Since the coarse-grained approach iterates over the attributes in a rule to match them with an example, we implemented a slightly different match functions for problems with different types of attributes. This helps keeping the code of the kernels as simple as possible in each case. If the problem has only continuous attributes a kernel for continuous attributes is used. If not, a kernel for mixed attributes is used. The latter includes an extra condition that checks if the attribute is nominal or real to perform the match calculations accordingly, which can potentially make the kernel slower. Moreover, the code used to match each attribute is exactly the same as

the code of the serial version.

Algorithm 8.3.2: REDUCE(*data*, *totalObjects*)

```

Determine the thread index tid and the global index gid
Declare input array in shared memory
[tid] ← 0
while gid < totalObjects
    do {
        if gid + blockSize < totalObjects
            then [tid] + = data[gid] + data[gid + blockSize]
            else [tid] + = data[gid];
        gid + = gridSize;
    }
SYNC()
if (tid < 256)
    then [tid] + = [tid + 256]
SYNC()
if (tid < 128)
    then [tid] + = [tid + 128]
SYNC()
if (tid < 64)
    then [tid] + = [tid + 64]
SYNC()
if (tid < 32)
    then {
        [tid] + = [tid + 32]
        [tid] + = [tid + 16]
        [tid] + = [tid + 8]
        [tid] + = [tid + 4]
        [tid] + = [tid + 2]
        [tid] + = [tid + 1]
    }
if (tid == 0)
    then {
        if (gridDim.x == 1)
            then Compact memory per classifier in global memory
        else Store data in global memory
    }

```

In [Bacardit and Krasnogor, 2009a] an efficient CPU-based match process for mixed attributes was proposed, which worked by separating the match of both types of attributes in two separate loops. This approach was also evaluated in our CUDA coarse-grained implementation. Our preliminary experiments showed that in CUDA it is possible to achieve more speedup by using only one loop that checks the attributes sequentially with an extra condition that checks the type of attribute instead of using two loops. This is because the divergent code of the *if* structure (which can slow the kernel by half) performs better than the non-coalesced memory access produced by the two loops. If two loops are used the memory access is not coalesced since the discrete attributes and the continuous attributes are interspersed in memory. The approach of reordering the memory of the classifier and instances to put together attributes of the same type is analysed in our fine-grained parallel approach, which will be explained in the next section.

8.3.2 Fine-grained parallel approach

In this approach we extended the parallelism of the previous approach to an extra dimension by parallelising the match of each one of the attributes in the rules. Since the ALKR representation allows to have discrete and continuous attributes at the same time, to parallelise the match of the attributes it is necessary to reorder them in memory by their type. This means to put

together discrete and continuous attributes during the structure flattening process. This is done over the instances and the classifiers. Moreover, the ALKR representation of the classifiers is expanded to create a classifier's structure of regular size. This means that in the CUDA structure all the attributes in the rule will be represented.

Since the attributes are separated by its type, the match is done independently for discrete and continuous attributes. Moreover, the class of the classifier is matched with the instance class in a third kernel. This means that instead of using only one kernel as in the previous approach, we are now performing the calculations with three simpler kernels.

Considering this, the fitness calculation using the fine-grained approach consists of four steps:

1. Compute the condition match for discrete attributes
2. Compute the condition match for continuous attributes
3. Compute the class match
4. Calculate the *and* between the condition match and the class match to determine the third variable and reduce the results per classifier iteratively.

Working with simpler kernels can potentially reduce the execution time by avoiding conditionals or non-coalesced memory accesses. However, expanding the original structures can also increase the execution time of the kernels since more memory will have to be read and processed. Since the computations are being performed independently in different kernels it is not possible to make a preliminary reduction to reduce the size of the output data as in the previous approach. In this approach the output data will be of size $x \cdot y \cdot A$ where A is 8 bytes as we need to store one 1 integer for the condition match and 1 integer for the class match per classifier-instance pair. Moreover, it is not necessary to have separate output structures for the discrete and continuous match kernels as these kernels are performed sequentially one after the other, and therefore, they can work over the same memory areas.

The following sections will explain how the ALKR representation is expanded to represent all the attributes in a rule. Afterwards, each one of the kernels of this approach will be explained further.

8.3.2.1 Expanding and reorganising the ALKR representation

To check discrete and continuous attributes independently and ensure coalesced memory access it is necessary to reorganise the memory structures where the classifiers and the instances are stored. Moreover, it is necessary to expand the representation of the rules so all the attributes are expressed. To do this we first determine which are the continuous and discrete attributes in the problem. In the case of the instances, no expansion is needed, so when creating the CUDA structures we first copy the values for all the discrete attributes, and then we copy the values of all the continuous attributes as shown in Figure 8.6.

In the case of the classifiers we also copy the discrete predicates first and then we copy the continuous predicates, copying first all the lower boundaries of the predicates and afterwards all the upper boundaries. In this new structure all the attributes will be represented, so for a discrete attribute that was not on the list before we include a *don't care* predicate (a predicate of all 1s), to ensure that this attribute will match any example. In the case of a continuous attribute that was not on the list before we copy the lower and upper bounds for the domain of that attribute. Moreover, after expanding the classifiers we still keep track of which were the attributes that were represented in the original lists using an extra structure of size equal to the number of attributes. If this structure expresses 1 in the byte corresponding to the attribute then the attribute was expressed in the original list. Otherwise, the attribute was not

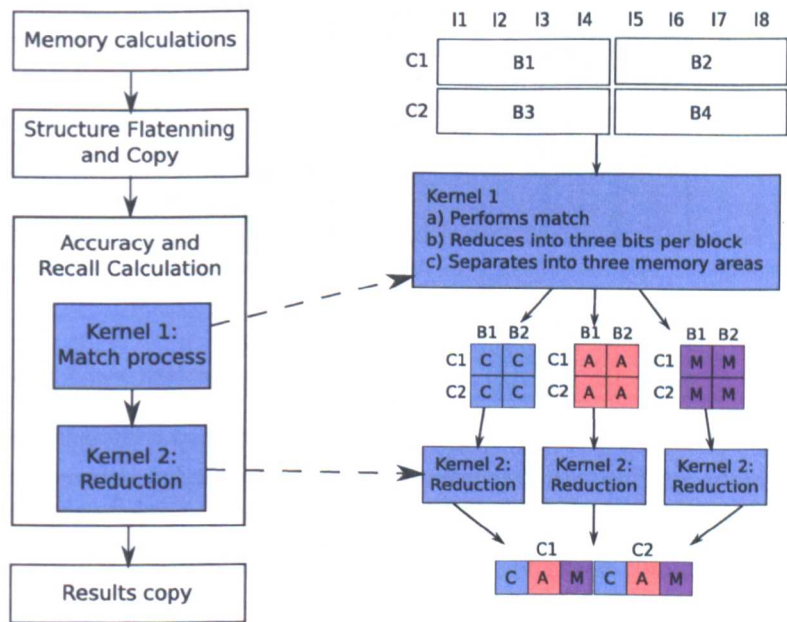


Figure 8.5: Complete flow of data for the coarse-grained parallel approach.

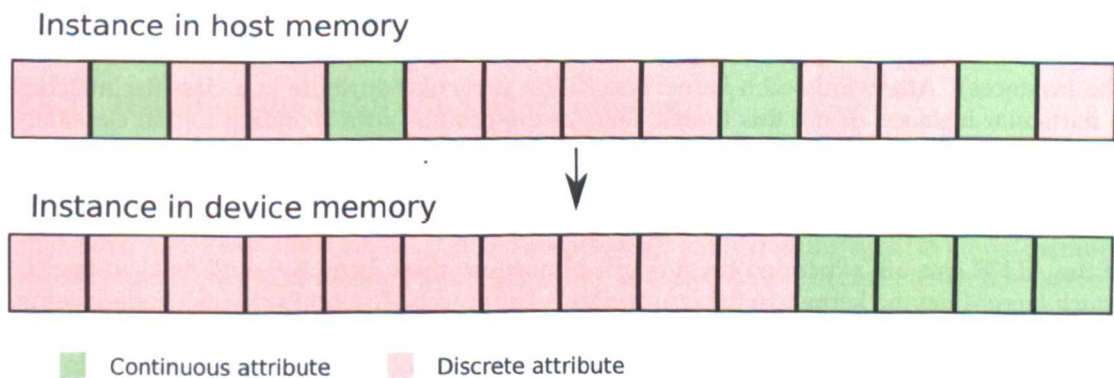


Figure 8.6: Example of reorganisation of an instance in device memory

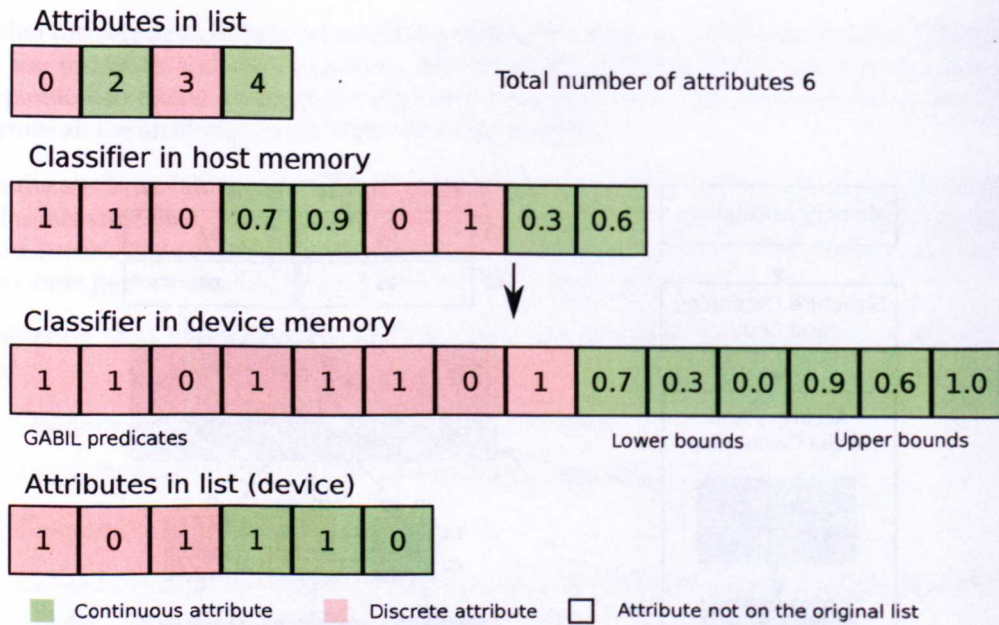


Figure 8.7: Example of reorganisation of a classifier in device memory

expressed. Figure 8.7 shows an example of how a rule would be expanded and reorganised in this approach.

8.3.2.2 Kernels 1 and 2: Discrete and continuous match

To match the attributes in this approach we launch one thread per classifier, attribute and instance. The configuration of both of the kernels is the same and it is the one that maximises the occupancy on the card. Each block will have 512 threads (8 threads for the attributes, 4 threads for the classifiers and 16 threads for the instances). In total the continuous kernel will execute $x \cdot y \cdot \text{numCon}$ threads while the discrete kernel will execute $x \cdot y \cdot \text{numDis}$, where numCon and numDis are the number of continuous and discrete attributes, respectively. In case the number of instances is very big and the system needs more blocks in the instance axis than the maximum allowed by the card,² the instances in this case are partitioned in groups which are checked sequentially.

First a matrix of size $x \cdot y$ is initialised with ones (assuming that all the classifiers match all the instances). Afterwards each kernel checks if a particular attribute in a classifier matches a particular instance. If not this thread changes the position corresponding to that classifier-instance pair to zero. If that particular position already had a zero the thread does not rewrite the memory. This allows both kernels to work with the same output structure sequentially, reducing the amount of memory that we need.

If the ALKR lists are expanded the amount of memory these kernels would need to read is much larger than the kernels in the coarse-grained approach. To stop the kernels for accessing memory corresponding to the attributes that were not expressed in the initial lists, these kernels only activate the threads corresponding to the relevant attributes in the original lists. To do this the threads access the additional structure created that shows if the attribute was relevant or not, and only activate the corresponding threads. The pseudocode for these kernels is shown in Algorithm 8.3.3.

²The maximum number of blocks allowed in one axis is 65535.

Algorithm 8.3.3: COMPUTEMATCH(*classifiers, instances, classActivation, numIns, numClass, numAtts, match*)

Determine the thread index T_{ijk} , the instance index I_j , the classifier index C_i and the attribute index A_k to access the global memory
 Declare *output* array in shared memory
 if $I_j < \text{numIns} \wedge C_i < \text{numClass} \wedge A_k < \text{numAtts}$
 then { if $\text{classActivation}[C_i][A_k]$
 then $\text{output}[T_{ijk}] \leftarrow \text{match}(\text{ins}[I_j].\text{atts}[A_k], \text{class}[C_i].\text{atts}[A_k])$
 else if $\text{output}[T_{ijk}] \leftarrow 0$
 Storing the final result in global memory
 if $\text{match}[I_j][C_i] \neq 0$
 then $\text{match}[I_j][C_i] = \text{output}[T_{ijk}]$

8.3.2.3 Kernel 3: Class match

The code for this kernel is very simple as each thread is in charge to compare the class of a particular classifier with the class of a particular instance. The resulting information at the end is stored in a matrix of size $x \cdot y$ which is independent from the output matrix of the previous kernel. The class match kernel launches 512 threads per block as well, 32 in the x-axis corresponding to the number of instances and 16 in the y-axis corresponding to the number of classifiers. In total this kernel launches $x \cdot y$ threads.

8.3.2.4 Kernel 4: Reduction

Since we have the results of the condition match and the class match in separate structures, the last kernel is in charge of performing the *and* operation to determine if a classifier matches both the condition and the class of a particular instance. After this value is calculated the reduction is performed in the same way as in the coarse-grained approach (see Section 8.3.1.2). In this particular reduction kernel we use a 1024 threads per block (256 threads on the x-axis for the instances and 4 threads in the y-axis for the classifiers). In this reduction we make the most out of the functionalities of the newest CUDA cards of 2.0 capability, which allow up to 1024 threads per block. This allows us to reduce the values of more instances at the same time, but also allow us the balance better the parallelism of the instances and the classifiers.

8.3.3 Handling the ILAS windowing scheme

In this chapter we also study the integration of our CUDA-based fitness computation and the ILAS windowing scheme, to verify if both efficiency-enhancement techniques can be combined efficiently. As we explained before, if all the instances left in the training set fit in global memory we copy them into global memory at the beginning of each GA run. In case the window system is activated, the CUDA fitness computation is called passing the offset of the window as an extra parameter. This is possible because the windows are created at the beginning of the GA run and all the windows are stored continuously in memory, hence it is just necessary to specify the offset to know which window the system is using.

Since the instances tend to occupy more memory than the classifiers, we save a lot of computational effort by copying them into global memory once per GA run and handling the windowing. However, when the GA finishes the instances that are covered by the new rule are removed from the training set and the windows are created again. At this point it is necessary to copy the instances again into global memory.

Also, better results were obtained when the large memory structures used (necessary to store instances and classifiers) were allocated only once at the beginning of the algorithm and were used over and over again rewriting the memory.

8.3.4 Integration with the learning algorithm

The CUDA evaluation is integrated inside BioHEL in two different stages: during the normal evaluation process of the whole population and during the elitism (where the best individuals of each window are kept in the population and reevaluated with the current window). Even though the latter only involves evaluating a few classifiers, preliminary experiments showed improvements in the performance when this part of the code was parallelised.

8.3.5 Handling GPGPUs with small global memory

Some GPGPUs might not have enough memory to store the whole problem at once in memory. Therefore, to consider this scenario at the beginning of the GA, it is necessary to calculate how many classifiers and instances fit in memory. If all the instances do not fit in memory, we calculate if it is possible to fit all the classifiers at once and revisit the instances in several iterations. If none of this is possible, the number of classifiers and instances that minimise the memory copy operations is calculated.

The heuristic we use to handle the worst case consists in assigning half of the memory for storing classifier information and the other half for storing instance information. To calculate this we solve a simple quadratic formula. So considering MI the memory occupied by each instance, MC the memory occupied by each classifier, A the size of the output structure used and MD the global device memory available, we try to calculate x the number of classifiers and y the number of instances to fit in memory.

For example in the coarse-grained approach the occupancy formula would look like:

$$MD = xMC + yMI + x \lceil \frac{y}{512} \rceil A \quad (8.3.2)$$

Moreover, we can assume $xMC = yMI$ and eliminate the ceiling by assuming the worst case as follows:

$$\lceil \frac{y}{512} \rceil < \frac{y}{512} + 1 \quad (8.3.3)$$

Substituting both equations in the Equation (8.3.2) we find out the we need to solve the following quadratic equation.

$$MD = x(2MC + A) + \frac{x^2 AMC}{512 MI} \quad (8.3.4)$$

In the case of the fine-grained approach we can follow the same procedure, but considering that there is no preliminary reduction on the first kernel. Therefore, the maximum output structure used is $x \cdot y \cdot A$ and the memory occupancy formula would look like:

$$MD = xMC + yMI + xyA \quad (8.3.5)$$

This produces an even simpler equation to solve, which is the following:

$$MD = 2 \cdot xMC + \frac{x^2 AMC}{MI} \quad (8.3.6)$$

Table 8.1: Datasets used for testing GPGPU-based fitness computation. $|T|$ = Training set size, $\#Att$ = Number of attributes., Cov = Coverage breakpoint

Continuos				Mixed				Discrete			
Name	$ T $	$\#Att$	Cov	Name	$ T $	$\#Att$	Cov	Name	$ T $	$\#Att$	Cov
sat	5790	36	0.1	adu	43960	14	0.01	CNbin	234638	10	0.001
wav	4539	40	0.1	far	90868	29	0.1	Par	235929	18	0.001
pen	9892	16	0.1	kdd	444619	41	0.1	c-4	60803	42	0.0025
SS	75583	300	0.0025	SA	493788	270	0.0025				
CN0	234638	20	0.001								
CN	234638	180	0.001								

8.4 Experimental design

To test the performance of our implementation we decided to perform two stages of experiments. First we evaluate the speedup of the evaluation process independently using the two approaches explained. Afterwards, we evaluate the speedup of the overall system after incorporating the CUDA evaluation process (using the best approach in the previous section) inside BioHEL.

For the first stage, we ran the evaluation process independently (both using the serial version and our two parallel approaches) by evaluating a random population 50 consecutive times (simulating a GA run but without selection nor exploration). We ran a complete GA per execution to test the advantage of copying the instances once at the beginning of each GA.

The speedup is compared over thirteen different problems explained in greater detail in Section 3.2. Table 8.1 contain a summarised list of the relevant characteristics of the datasets regarding our analysis, along with the coverage breakpoint used (necessary for replication purposes). For each problem, the original datasets were partitioned for ten-fold cross validation. The training set size shown in Table 8.1 is the size of each one of the folds. Over each fold we ran the evaluation process with 25 different seeds. To determine the impact of combining the ILAS windowing scheme with the proposed evaluation process we did experiments with different number of windows $w = \{1, 2, 4, 6, 8, 10, 15, 20, 25, 30, 35, 40, 45, 50\}$. The rest of BioHEL's parameters remain the same as the base configuration shown in Section 3.3, except for the Initial MDL Theory Length ratio which was set to 0.25.

For the second stage of experiments we compare the whole learning process of BioHEL using the CUDA evaluation process integrated with the serial version of BioHEL used throughout this thesis. In this stage we only use the parallel approach that obtained the best results in the previous stage. We used the same problems as in the previous stage, with the addition of *CNbin* and *CN0*, and we performed experiments with $w = \{1, 5, 10, 25, 20, 25, 30, 35, 40, 45, 50\}$. Because of the computational cost of each experiment, we ran each one of the folds only once in this stage.

In the last section we present empirical models that explain the execution times obtained with the approach that obtains the best results. These model are based on the training set size, number of attributes in the problem and the number of windows used.

For the CUDA experiments we used two different architectures, which we intend to compare. First, we used a Pentium 4 of 3.6GHz with hyper-threading, 2GB of RAM and a Tesla C1060 with 4GB of global memory and 30 multiprocessors. Over this architecture only the coarse-grained approach was tested which was the one implemented earlier in time [Franco et al., 2010a]. For our most recent experiments which include the fined-grained and the coarse-grained parallel approaches we used a newer hardware which consist of an Intel(R) Core(TM) i7 CPU of 8 cores at 3.07GHz and 12GB of RAM with a Tesla C2070 with 6GB of internal memory and 448 CUDA cores (14 multiprocessors \times 32 CUDA Cores/MP). Moreover, for the serial ex-

periments we used the High Performance Computing facility of the University of Nottingham each node with 2 quad-core processors (Intel Xeon E5472 3.0GHz). For this experiments we wanted to compare our implementation against the most likely architecture a user would use if they do not have access to the GPGPU technology.

Along the analysis the speedup of the CUDA evaluation process over the serial algorithm is reported in order to determine the advantage of using the parallel architecture. For interpretation and replication purposes we also report the execution time of the baseline cases (using 1 window). The accuracy of the system is not reported since both implementations behave identical and obtain the same accuracy when run with the same random seed.

Table 8.2: Execution time in seconds of the evaluation process of the serial version and both CUDA fitness functions using 1 window

	GPU Model	Serial		Coarse-grained		Fine-grained
		–		C1060	C2070	C2070
Cont.	sat	3.6±	0.2	1.9±0.0	0.6±0.0	0.84±0.01
	wav	2.6±	0.1	1.6±0.0	0.4±0.0	0.70±0.01
	pen	4.9±	0.2	2.2±0.0	0.5±0.0	0.70±0.01
	CN0	449.2±	49.1	–	11.7±0.2	17.18±0.08
	CN	1555.9±	452.8	42.4±0.6	24.1±0.4	59.93±0.53
	SS	770.6±	119.5	14.7±0.2	9.0±0.2	22.89±0.30
Mixed	adu	147.9±	30.9	10.4±0.1	2.6±0.0	2.59±0.01
	kdd	1715.7±	632.4	95.9±1.4	102.4±1.7	51.66±0.20
	far	420.8±	90.6	23.1±1.0	8.7±0.2	8.69±0.02
	SA	3776.4±	1212.8	90.5±1.2	55.0±1.0	144.81±1.30
Dis.	c-4	343.8±	71.9	17.9±0.2	12.8±0.2	7.07±0.03
	Par	863.7±	163.1	60.0±0.6	18.0±0.2	15.74±0.04
	CNbin	544.5±	150.9	–	8.6±0.1	11.16±0.05

8.5 Performance of the evaluation process

In Table 8.2 we present the execution times for the evaluation process using one window of the serial version and CUDA versions (the coarse-grained and fine-grained approach). This table shows that the variance in the execution time of the CUDA algorithm is always smaller than the serial one, which indicates that the average speedup reported later is statistically significant. Moreover, for the coarse-grained approach we show the execution times for the former architecture used in Franco et al. [2010a], the Tesla C1060, and the newest architecture, the Tesla C2070. It is worth mentioning that the problems *CN0* and *CNbin* were added to the dataset pool later in time, when the former architecture was not available anymore. This is the reason why results using the C1060 over these problems are not reported. However, it was interesting to add these datasets since they represent problems characteristics that we had not explore before.

Moreover, Figure 8.8 shows the speedups corresponding to the information in Table 8.2. In this figure is it noticeable that even when both parallelisation approaches produce speedups with respect to the serial approach, the coarse-grained approach produces more speedup than the fine-grained approach in 5 problems (*CN0*, *CN*, *SS*, *SA*, *CNbin*), equal execution times in 5 problems (*sat*, *pen*, *wav*, *Adu*, *far*), and lower speedup in only 3 problems (*c-4*, *kdd*, *Par*). Particularly in the *kdd* problem the fine-grained approach doubles the speedup of the coarse-grained approach. This could be do to the fact that the *kdd* problem has a balanced number of continuous and discrete attributes which will distribute the computation of both match kernels evenly. However, in general when the problem is big, the coarse-grained approach seems to perform better.

Table 8.3: Speedup of the CUDA evaluation using the coarse-grained approach (on the C1060 and C2070) compared to the serial version using the same window (first row) and the serial version without using windowing (second row)

		Number of Windows													
		1	2	4	6	8	10	15	20	25	30	35	40	45	50
Coarse-grained Tesla C1060															
Cont. problems	sat	1.9	1.3	0.8	0.6	0.5	0.4	0.3	0.2	0.2	0.2	0.2	0.1	0.1	0.1
		1.9	2.5	2.8	2.9	2.9	3.0	3.1	3.1	3.1	3.1	3.1	3.1	3.1	3.1
	wav	1.6	1.1	0.6	0.4	0.3	0.3	0.2	0.2	0.1	0.1	0.1	0.1	0.1	0.1
		1.6	1.9	2.1	2.1	2.1	2.2	2.2	2.2	2.2	2.2	2.2	2.2	2.2	2.2
	pen	2.2	1.7	1.1	0.7	0.6	0.5	0.4	0.3	0.3	0.2	0.2	0.2	0.2	0.2
		2.2	3.1	3.7	3.6	3.5	3.6	4.1	4.2	4.2	4.2	4.2	4.2	4.2	4.2
	CN	36.7	43.1	40.3	39.8	37.3	36.1	31.6	27.3	22.7	18.7	16.7	14.7	13.4	12.8
		36.7	69.9	128.4	178.3	220.7	258.4	327.8	389.3	432.9	463.3	488.7	506.8	522.3	536.7
SS	52.4	52.5	50.0	46.1	38.6	30.3	15.2	12.1	10.2	8.4	8.2	8.3	8.5	8.6	
	52.4	96.9	168.0	222.7	264.1	293.7	344.7	374.8	394.3	406.9	416.8	424.4	430.0	435.2	
Mixed problems	adu	14.2	15.8	7.7	3.5	3.0	2.9	2.6	2.4	2.4	2.3	2.2	2.1	2.1	2.0
		14.2	25.3	41.6	52.9	61.5	68.0	78.4	84.3	88.0	90.8	92.2	93.2	94.5	95.1
	Par	14.4	18.3	22.5	23.1	22.9	21.8	16.6	10.2	7.6	6.9	6.5	6.4	6.2	6.0
		14.4	27.8	51.7	72.7	91.0	107.2	141.1	168.4	189.5	208.3	222.9	235.1	245.1	255.6
	kdd	17.9	23.1	26.3	27.6	27.9	28.2	27.0	26.4	24.6	22.4	19.7	16.7	14.8	13.1
		17.9	34.9	65.3	91.9	115.1	138.3	182.5	218.9	248.1	272.7	291.7	311.4	324.8	339.7
	far	18.2	23.4	24.0	18.9	12.8	8.6	5.2	4.8	4.7	4.9	5.0	5.0	4.9	4.9
		18.2	33.7	60.4	81.3	99.6	115.8	144.6	164.5	176.0	186.4	191.2	197.0	200.9	203.5
	c-4	19.2	24.1	24.8	18.1	9.7	6.6	5.0	4.5	4.4	4.3	4.5	4.7	4.7	4.4
		19.2	35.6	63.3	86.4	104.5	120.0	146.5	162.7	173.1	180.6	184.9	189.4	192.1	194.1
	SA	41.8	35.9	37.0	35.8	35.5	34.7	33.3	32.1	31.0	29.6	28.6	27.5	26.0	24.5
		41.8	79.5	146.2	203.7	252.8	296.3	383.7	449.0	502.3	542.7	578.0	605.8	630.4	649.9
Coarse-grained Tesla C2070															
Cont. problems	sat	6.3	5.2	3.9	3.1	2.7	2.3	1.9	1.6	1.4	1.2	1.1	0.9	0.9	0.9
		6.3	9.7	13.8	15.7	17.1	18.0	20.0	21.2	21.2	21.2	21.2	21.2	21.2	22.5
	wav	5.8	4.7	3.3	2.5	2.0	1.9	1.5	1.1	0.9	0.8	0.8	0.8	0.7	0.6
		5.8	8.6	11.2	12.2	12.8	14.3	15.1	15.1	15.1	15.1	16.1	16.1	16.1	16.1
	pen	10.1	8.6	6.1	4.9	4.0	3.5	2.7	2.4	2.1	1.8	1.6	1.5	1.4	1.3
		10.1	15.4	20.6	23.5	24.7	26.0	27.4	30.9	30.9	30.9	30.9	30.9	30.9	32.9
	CN0	38.5	57.8	66.8	65.6	61.5	55.6	35.1	21.8	18.7	18.7	18.8	19.0	19.2	19.5
		38.5	73.9	137.8	193.6	242.8	286.1	380.7	453.8	510.5	561.5	599.0	632.7	660.6	691.1
CN	64.6	77.3	74.5	75.6	72.9	71.8	67.0	58.6	50.9	43.7	39.6	36.2	33.4	32.8	
	64.6	125.3	237.2	339.0	431.0	513.5	694.6	836.5	972.4	1080.5	1161.1	1244.7	1307.5	1376.9	
SS	85.7	89.6	92.2	90.1	80.4	67.3	38.1	33.5	30.6	27.1	27.5	28.4	29.3	30.5	
	85.7	165.4	309.5	435.4	550.4	653.1	865.9	1041.4	1185.5	1306.1	1401.1	1454.0	1481.9	1541.2	
Mixed problems	adu	57.5	65.0	32.8	15.1	13.1	12.8	12.0	10.9	11.4	10.8	10.8	10.4	10.5	10.1
		57.5	104.1	176.0	227.5	268.8	301.8	360.6	389.1	422.5	434.9	448.1	462.1	477.0	477.0
	Par	48.1	61.0	75.5	78.3	78.1	74.7	57.8	35.5	27.0	24.4	23.1	22.7	22.3	21.6
		48.1	92.7	173.8	246.1	310.7	367.5	490.8	587.6	669.6	732.0	792.4	838.6	881.4	918.9
	kdd	16.8	21.8	25.8	28.0	29.3	29.9	30.8	31.4	30.6	28.8	26.3	23.0	21.0	18.9
		16.8	32.9	63.9	93.3	120.7	147.0	207.7	260.7	308.6	350.9	389.0	427.8	461.2	490.2
	far	48.1	63.8	67.5	54.7	37.5	25.4	15.6	14.7	14.9	15.7	16.4	16.8	16.6	16.9
		48.1	92.1	169.7	235.1	292.2	342.1	433.8	500.9	553.7	592.6	628.0	657.5	678.7	701.3
	c-4	26.8	36.3	42.3	34.9	20.2	14.1	11.4	10.6	10.7	10.9	11.7	12.6	12.8	12.2
		26.8	53.7	107.8	166.9	217.6	258.5	330.5	381.9	424.4	458.3	484.2	505.5	520.8	537.1
	SA	68.7	60.1	63.8	62.9	63.7	63.4	62.8	62.6	61.6	60.3	58.7	57.2	54.7	52.2
		68.7	133.3	252.1	357.6	453.9	540.3	724.8	874.2	999.0	1104.2	1187.5	1258.8	1325.0	1388.4
CNbin	63.3	98.8	119.5	120.0	116.4	108.8	71.2	40.0	30.4	27.2	27.1	27.0	27.9	27.0	
	63.3	119.9	221.3	307.6	383.5	450.0	585.5	680.6	766.9	825.0	878.3	922.9	972.4	1008.4	

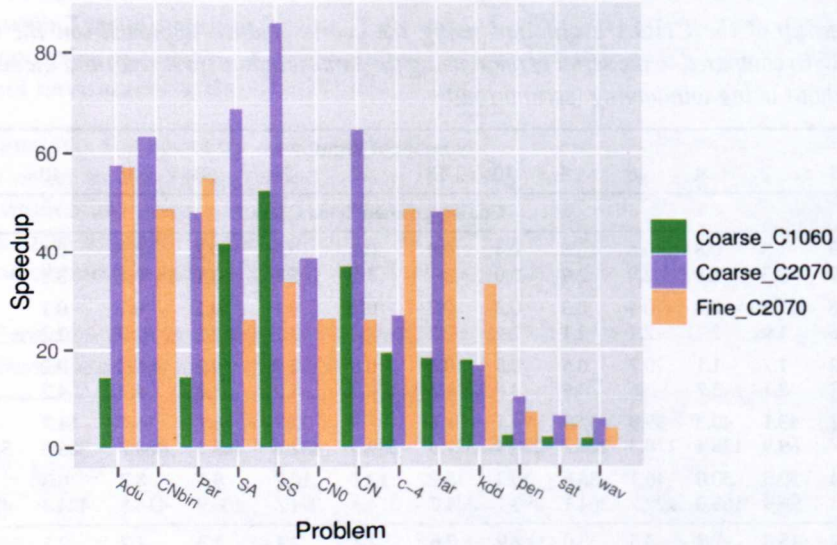


Figure 8.8: Speedup against the serial algorithm without using windowing of the different parallelisation approaches ran on different architectures

Regarding the usage of windows, in Table 8.3 we present the net speedup (speedup of the CUDA evaluation against the serial version using the same window size) and the total speedup (speedup of the CUDA evaluation against the serial version without using windowing) for the coarse-grained approach using different GPGPU architectures. In this table we can notice that the coarse-grained parallel evaluation process achieves speedups for all the problems when not using windowing (using one window). Moreover, the speedup gets considerably higher in the problems with more than 40000 instances. Also we can observe that the speedups vary depending on the GPGPU used. The highest net speedup is achieved in the SS problem, where with the Tesla C1060 we achieved 52.4X and with the Tesla C2070 we achieved 85.7X. Combined with the ILAS windowing scheme the highest total speedup observed using the C1060 is 649.9X in the SA problem. On the other hand, using the C2070 the highest total speedup is 1541.2X in the SS problem. In general, the speedups using the C2070 are larger than the ones using the C1060, which is expected since this GPGPU possesses more cores. For small problems, it seems like the speedup increases between 4 or 5 times with the use of a newer architecture, while for larger problems the increase in the speedup seems to be problem dependant.

Table 8.4 shows the corresponding speedup results of the evaluation process using the fine-grained approach on the Tesla C2070. The same as in the previous table, we can see that this approach also obtains speedups for all problems, but the speedups follow a different pattern than the one observed in the previous table. The largest net speedup is found in the Adu problem, which obtains 57.1X speedup. Moreover, the highest total speedup found combining the ILAS windowing and the CUDA fitness function is of 949.2X in the Par problem.

In general in the two previous tables, we can also observe that the speedup varies with the number of windows and for a higher number of windows the net speedup tends to decrease. This is because the CUDA evaluation gets advantage of using large or medium size training sets, since all the example comparisons are done in parallel. The same happens with the problems that are very small from the start (pen, sat and wav). In these datasets, the overhead of parallelising the fitness function is sometimes greater than the advantage obtained when using large windows. However, for small problems large windows should not be used (and neither they are necessary) because, as discussed in Chapters 2 and 4, this produces a higher chance of problem disruption.³

³Each window used is not a reliable partition of the problem to solve

Table 8.4: Speedup of the CUDA evaluation using the fine-grained approach (on the C2070) compared to the serial version using the same window (first row) and the serial version without using windowing (second row)

		Number of Windows													
		1	2	4	6	8	10	15	20	25	30	35	40	45	50
		Fine-grained Tesla C2070													
Cont. problems	sat	4.3	3.7	2.8	2.3	2.0	1.7	1.4	1.1	1.0	0.9	0.8	0.7	0.7	0.6
		4.3	6.9	10.0	11.6	12.9	13.3	14.4	15.0	15.7	15.7	16.4	16.4	16.4	16.4
	wav	3.7	3.2	2.3	1.9	1.5	1.4	1.0	0.8	0.7	0.6	0.6	0.5	0.5	0.5
		3.7	5.7	7.8	9.2	9.5	10.3	10.7	11.2	11.2	11.7	11.7	11.7	11.7	11.7
	pen	7.1	6.3	4.6	3.6	3.1	2.8	2.2	1.9	1.6	1.4	1.3	1.2	1.1	1.0
		7.1	11.2	15.4	17.6	19.0	20.6	22.5	23.5	23.5	23.5	24.7	24.7	24.7	24.7
Mixed problems	CN0	26.1	39.5	46.0	45.3	42.6	38.8	24.4	15.1	13.1	13.0	13.1	13.4	13.6	13.8
		26.1	50.5	94.8	133.7	168.3	199.7	264.3	314.1	356.5	390.6	419.8	444.8	467.9	488.3
	CN	26.0	31.2	30.1	30.5	29.4	29.1	27.1	23.8	20.4	17.5	15.9	14.5	13.4	13.1
		26.0	50.5	95.9	136.8	174.0	208.3	280.3	340.5	389.9	432.2	467.2	498.7	525.6	549.8
	SS	33.7	35.2	36.2	35.3	31.4	26.2	14.8	12.8	11.6	10.3	10.4	10.9	11.5	12.1
		33.7	65.0	121.4	170.9	215.3	254.3	335.0	399.3	450.6	494.0	527.8	558.4	583.8	611.6
	Adu	57.1	63.3	30.9	14.2	12.3	11.8	10.7	9.7	10.0	9.4	9.4	9.0	9.0	8.7
		57.1	101.3	166.1	214.3	250.6	279.0	321.4	343.9	369.6	379.1	389.1	399.6	410.7	410.7
	Par	54.9	69.4	85.5	87.8	86.8	82.4	62.7	38.1	28.5	25.7	23.9	23.6	23.0	22.3
		54.9	105.3	196.7	276.0	345.5	405.5	533.2	630.5	708.0	771.2	822.6	872.5	909.2	949.2
Mixed problems	kdd	33.2	42.8	49.4	52.4	53.7	53.7	52.4	51.3	48.3	44.2	39.2	33.2	29.5	26.1
		33.2	64.7	122.5	174.5	221.1	263.5	353.7	425.7	487.4	537.8	579.6	617.1	649.9	678.1
	far	48.4	63.3	64.9	51.8	34.6	23.1	14.1	13.1	13.2	13.9	14.2	14.5	14.3	14.5
		48.4	91.5	163.1	222.6	269.7	311.7	393.3	447.6	489.3	526.0	546.5	568.6	584.4	601.1
	c-4	48.6	61.8	64.8	47.3	25.5	17.2	13.7	12.7	12.7	12.8	13.7	14.8	14.8	14.2
		48.6	91.4	165.3	226.2	275.0	315.4	399.7	458.3	505.5	537.1	563.5	592.7	603.1	625.0
	SA	26.1	22.9	24.4	24.2	24.5	24.5	24.4	24.5	24.2	23.7	23.3	22.8	21.8	20.8
		26.1	50.8	96.3	137.4	174.7	208.8	282.0	342.1	391.7	434.6	470.9	501.5	528.9	553.7
Mixed problems	CNbin	48.8	77.2	93.3	94.0	90.8	85.0	55.2	31.0	23.4	21.1	21.0	21.0	21.4	20.8
		48.8	93.7	172.9	240.9	299.2	351.3	453.8	528.7	591.9	640.6	680.6	716.5	745.9	777.9

To analyse better the relation between the speedup obtained with different windows and the characteristics of the problems (i.e. number of attributes, type of problem, training set size), Figure 8.9 shows the relation between training set size and net speedup (on the left), and the relation between the number of windows and the total speedup (on the right) for each one of the problems and implementations analysed. For the different number of windows w we consider the training set size T_w to be equal to the size of the strata ($T_w = |T|/w$). We observe that the relationship between the speedup and the training set is not linear and the behaviour of the speedup changes depending on the type of problem and architecture used. Between 25000 and 50000 instances the speedup curves get their maximum value in most of the problems and afterwards they get steady or decrease. A possible explanation for the decrease in the speedup when the training set gets bigger is that the number of blocks needed to launch the match kernel depends on the size of the problem. This produces an increase of the size of the output structure copied into global memory at the end of this kernel. Moreover, there is a limit in the number of blocks that can be processed in parallel. From a certain value, the blocks are served sequentially. Iterative strategies to process more than one instance in a thread could be analysed to avoid launching a large number of blocks.

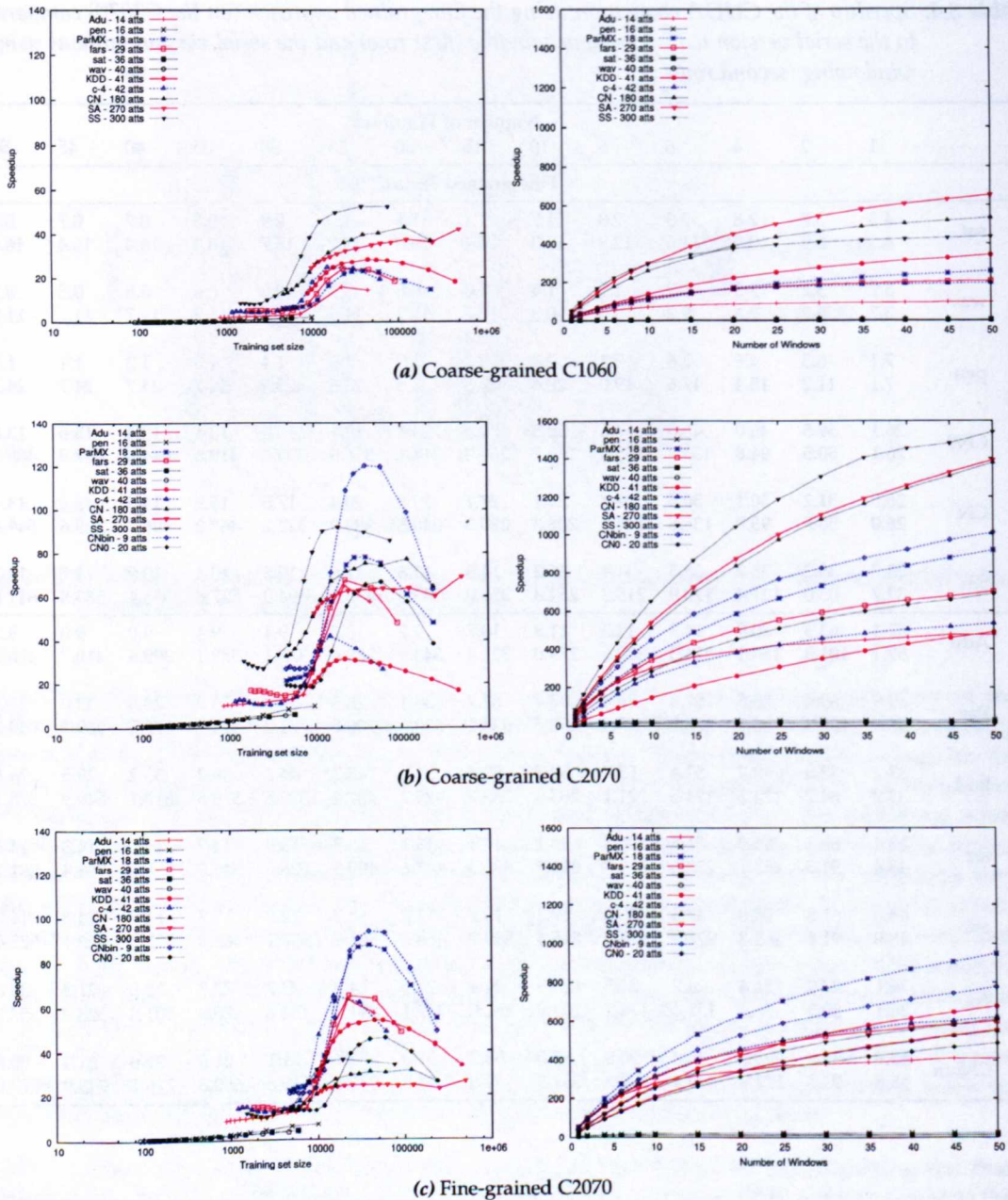


Figure 8.9: Net speedup against the training set size (left) and total speedup according to the number of windows (right) of the independent evaluation process using different approaches and architectures. Problems: Black = Continuous, Red = Mixed, Blue = Discrete.

Moreover, the steep decrease in the speedup when the training set gets very small is because up to this point the serial algorithm can fit the examples in cache memory. Also the usage of CUDA when the training set is small is less beneficial, because the constant overhead produced by the memory copy operations cannot be hidden by the speedup gained.

The patterns of the coarse-grained approach in Figure 8.9 seem to be similar for both architectures tested, except for the *Par* problem which seems to gain considerably more speedup in the C2070. However, the patterns of the fine-grained approach are considerably different than the patterns of the coarse-grained approach. While for the coarse-grained approach using the C2070, the continuous and the discrete problems seem to get the largest speedups, for the fine-grained approach only the discrete problems seem to get the largest speedups, followed by the

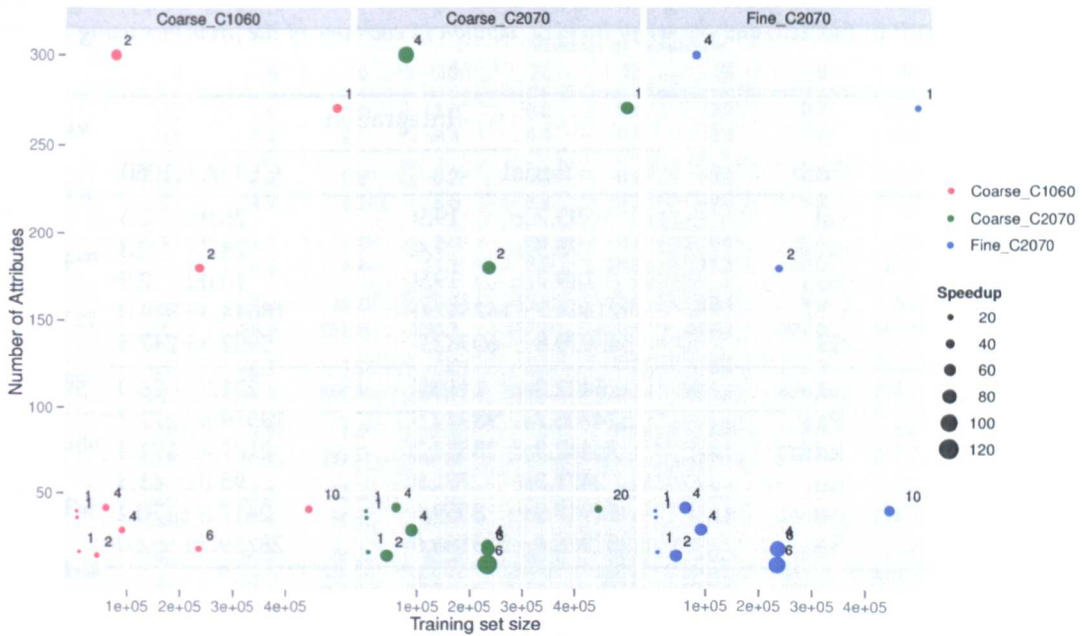


Figure 8.10: Largest speedup found for each one of the implementations and different architectures tested depending on the number of attributes and training set size. The size of the circle represents the speedup and the number besides it represents the window used to get that particular speedup.

mixed ones. Moreover, the fine-grained approach seems to obtain higher speedups in problems with less attributes.

Regarding the total speedup we can see it increases logarithmically with the number of windows. Moreover, the patterns observed among the coarse-grained approach with both architectures and the fine-grained approach are completely different. While in the coarse-grained approach the problem with more continuous attributes obtain the largest speedups, in the fine-grained approach the largest speedups are found in discrete problems with few attributes.

Figure 8.10 shows the maximum speedup obtained according to the number of attributes and training set size. In this figure we can see that the coarse-grained implementation seems to benefit problems with large number of attributes, while the fine-grained approach seems to obtain better results in problems with less attributes. Moreover, in both implementations we can observe that the problems with medium training set sizes obtain the best results, and these results are usually obtained when using a couple of windows. In particular the coarse-grained approach obtains better results than the fine-grained approach in large problems. This is due to the fact that the number of cores in a GPGPU is also limited and the best results are found when we can find the right balance between parallelism and thread workload.

Moreover, the difference in the speedup patterns among problems is given also due to the fact that the number of blocks that we launch is dependant on the characteristics of the problem. In the case of the coarse-grained approach it is dependant only on the training set size, but in the case of the fine-grained approach it is also dependant on the number of attributes. Some block configurations depending on the architecture used might be more beneficial than others, which explains the differences in the speedup among problems.

Table 8.5: Execution time (s) of the integration of the CUDA fitness function using the coarse-grained approach in BioHEL and the serial BioHEL version in each one of the problems using 1 window

	Prob.	Integration	
		Serial	CUDA C1060
Cont.	sat	95.7± 19.9	25.9± 2.5
	wav	75.5± 9.4	24.7± 0.8
	pen	149.7± 19.9	40.0± 2.9
	CN	821464.7±167542.0	18644.3±944.0
	SS	347979.8± 60982.7	5992.3±247.5
Mixed	adu	5422.8± 1410.7	271.7± 26.0
	Par	524706.7± 98949.5	19559.8±671.7
	kdd	76442.3± 23533.2	2102.4±191.3
	far	2471.3± 701.8	95.0± 41.5
	c-4	52917.9± 8059.6	2417.8±170.2
	SA	1252976.8±203186.6	28759.7±552.0

In general we can conclude that the coarse-grained approach obtains better results in terms of large scale datasets because it produces higher speedups for problems with large training set sizes and large number of attributes. Moreover, this is the approach that produces the highest cumulative speedups. Even though the speedup varies depending on the window size, problems parameters and configuration parameters (i.e number of iterations per GA), we have shown that the coarse-grained methodology presented improves the performance of the evaluation process when the training set is sufficiently big.

In the next section we are going to present the results of the performance of BioHEL after integrating the CUDA-based fitness function using the coarse-grained approach. Later we will analyse this approach more in deep generating models that explain its execution times.

8.6 BioHEL using CUDA-based evaluation

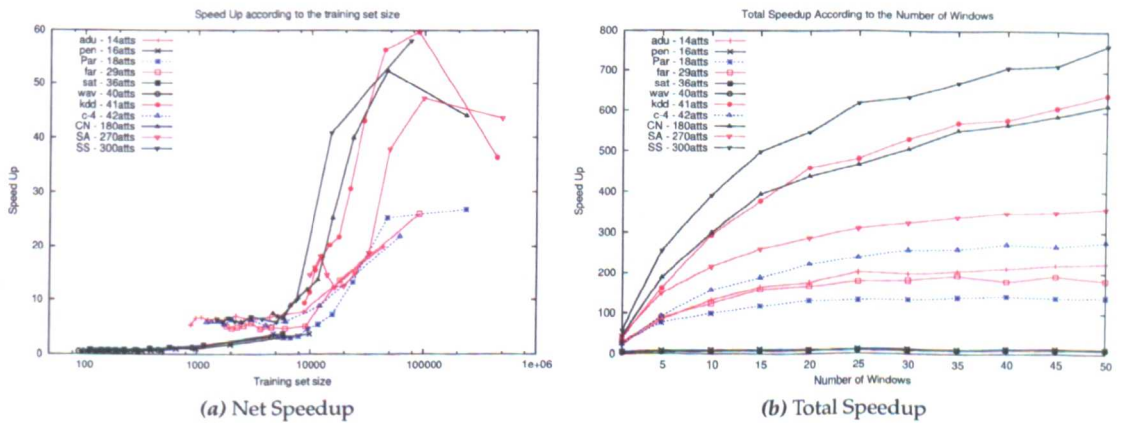
Table 8.5 shows the execution times of the base case (1 window) of the whole learning process using both the CUDA version and the serial version of BioHEL. In this section the results obtained in [Franco et al., 2010a] are presented, which were run in the Tesla C1060 and do not include the problems *CNbin* and *CN0*. However, we believe that the results in newer architectures will follow similar patterns as the ones presented in this section.

Table 8.6 presents the speedup of the BioHEL system using the CUDA evaluation over the serial version. In this table we can notice that the speedup increases compared to the results of the raw evaluation. This is due to the usage of the IRL approach. While the previous experiments were done with the full training set, using IRL each time we learn one rule the training set size decreases. When the training set gets slightly smaller the CUDA evaluation might obtain more speedup as it was shown in the previous section. Nevertheless, there is also a downside, when the training set is too small the overhead of performing the necessary memory copy operations overcomes the advantage obtained by using a GPGPU. This aspect is not considered in the experiments of the previous stage. Using a large number of strata amplifies this phenomena because the system performs less calculations in parallel but the memory copy overhead remains constant.

Using the integrated version of BioHEL with the GPGPU-based evaluation process, the SS

Table 8.6: Speedup of the BioHEL system using the CUDA evaluation over the serial version

		Number of Windows										
		1	5	10	15	20	25	30	35	40	45	50
Cont. problems	sat	3.7	1.6	1.2	1.0	0.8	0.8	0.7	0.7	0.7	0.6	0.8
		3.7	5.6	6.3	8.4	8.3	9.5	8.2	9.4	9.7	8.8	10.6
	wav	3.1	1.2	0.8	0.7	0.7	0.6	0.6	0.5	0.6	0.5	0.5
		3.1	4.7	6.2	8.6	8.9	9.3	9.3	9.2	9.9	9.4	9.3
	pen	3.7	1.7	1.0	1.0	0.7	0.7	0.6	0.6	0.5	0.6	0.5
		3.7	8.1	8.6	11.1	9.8	11.3	11.3	10.6	10.8	11.3	11.5
	CN	44.1	52.4	40.0	25.3	13.8	11.9	10.4	9.0	6.7	7.0	7.6
		44.1	188.9	298.8	394.1	437.0	466.4	504.2	552.5	565.8	588.0	615.5
	SS	58.1	40.9	9.9	6.0	6.6	7.0	6.0	6.1	6.8	6.1	6.6
		58.1	256.0	390.9	498.7	546.1	620.1	635.3	671.9	708.5	714.0	765.3
Mixed problems	adu	20.0	7.9	6.9	6.4	7.1	6.3	6.5	6.4	6.8	6.7	5.4
		20.0	83.8	132.0	165.2	176.3	201.1	198.1	204.9	211.2	218.1	224.3
	Par	26.8	25.3	13.4	7.4	5.6	4.7	3.2	2.9	2.9	3.0	3.6
		26.8	77.6	99.8	118.6	130.7	133.0	133.8	140.3	141.7	138.2	139.9
	kdd	36.4	59.7	56.3	43.2	30.7	21.7	20.3	18.1	15.6	11.4	9.4
		36.4	161.7	292.7	376.7	457.7	480.8	528.9	571.4	578.6	608.6	642.3
	far	26.0	13.7	5.2	4.7	4.9	4.6	5.7	5.2	4.9	4.6	5.1
		26.0	89.1	123.4	159.2	166.3	178.7	181.9	194.7	180.1	192.0	182.0
	c-4	21.9	8.9	6.1	5.3	6.4	5.9	6.4	5.5	6.0	6.0	5.9
		21.9	92.9	157.1	188.8	220.9	238.7	257.4	259.9	271.0	265.3	277.5
	SA	43.6	47.3	37.9	18.8	15.0	12.6	12.4	14.7	18.1	16.0	14.7
		43.6	148.9	214.2	259.6	285.3	309.1	322.8	338.1	347.6	350.2	359.3

**Figure 8.11:** Net speedup against the training set size (left) and total speedup according to the number of windows (right) of the integrated learning process. Black = Continuous, Red = Mixed, Blue = Discrete.

dataset obtained the best results, both without combining it with the ILAS windowing scheme (with a maximum speedup of 58.1X) and in combination with it (with a maximum speedup of 765.3X). Figure 8.11 shows the relation between the training set size and speedup of the integrated system (on the left) and the total speedup of the learning process according to the number of windows (on the right). We can observe similar behaviours in the speedup as the ones explained in the previous section. In the case of the integration the maximum speedup for most of the problems can be found around 50000 and 100000 examples. This shift is expected because as some examples are deleted and the training set becomes smaller, its size gets closer to the evaluation “sweetspot” that increases the speedup. In most datasets, specially the large ones, we can observe that as the number of windows increases, the total speedup (compared

Table 8.7: Table of fitted models for the match and the instance copy processes where x is the number of attributes and y is the number of instances in the training set.

	Function	Model	p-value	R^2
C2070	Dis. Match	$2.689e^{-3} - 1.015e^{-3} \cdot \log(x) - 1.072e^{-6} \cdot y + 8.864e^{-7} \cdot \log(x) \cdot y$	$< 2.2e^{-16}$	0.999
	Cont. Match	$8.333e^{-4} - 2.459e^{-4} \cdot \log(x) - 1.713e^{-7} \cdot y + 3.729e^{-7} \cdot \log(x) \cdot y$	$< 2.2e^{-16}$	0.997
	Ins. Copy	$2.132e^{-4} + 1.302e^{-6} \cdot x + 2.275e^{-9} \cdot y + 7.989e^{-10} \cdot y \cdot x$	$< 2.2e^{-16}$	0.999
C1060	Dis. Match	$4.229e^{-6} \cdot y$	$< 2.2e^{-16}$	0.998
	Cont. Match	$3.655e^{-6} \cdot y$	$< 2.2e^{-16}$	0.974
	Ins. Copy	$1.912e^{-4} + 1.753e^{-9} \cdot y + 7.021e^{-10} \cdot x \cdot y$	$< 2.2e^{-16}$	0.997

to the serial non-windowed BioHEL) also increases. Thus, we have shown that both efficiency-enhancement mechanisms can be successfully combined. However, the problems that give the largest speedups using one window, might not be the ones that obtain the largest speedup overall. It is noticeable that large problems in terms of number of instances such as SA and Par increase their net speedup when using a couple of windows instead of just one.

In this section we learned that the GPGPU evaluation process can be successfully integrated in BioHEL obtaining speedups up to 765X when combining this methodology with the ILAS windowing scheme. Moreover, the behaviour of the speedup of the whole learning process follows the same patterns observed in the evaluation process. These results push the experimental boundaries of BioHEL since as we can see for one of the largest problems (SA) the execution times were reduced from two weeks to eight hours.

8.7 CUDA execution time models

In order to understand better the results obtained in the two previous sections we generated models to explain the execution times obtained with the coarse-grained approach in the two architectures analysed. To do this we ran the evaluation process of the coarse-grained approach independently with problems that vary their number of attributes and the training set size in a regular step, to sample uniformly the whole problem space.

For the match kernel we generated independent models for discrete and continuous problems. For the discrete problems we used the k-DNF family of problems with $d = \{10, 11, 12, 13, 14, 15, 20, 60, 100, 140, 180, 220, 260, 300\}$ and training set size varying from 1048576 instances to 256 instances.⁴ For the continuous problems we used the complete PSP benchmark dataset which can be found in <http://icos.cs.nott.ac.uk/datasets/psp/download.html> and which have a number of attributes $d = \{20, 60, 100, 140, 180, 220, 260, 300\}$. To use these datasets we sampled the training set to create samples from 234638 instances to 256 instances. CN0 and CN are part of this benchmark set.

In preliminary experiments we realised that the operations that took most of the time were the copy of the instances from host to device and the match kernel. Therefore, for simplicity the models will only consider these 2 steps.

Figure 8.12 shows the fitted model of the execution time of the match kernel in seconds according to the training set size and the number of attributes. The formulas corresponding to the models are shown on Table 8.7. This figure shows that for the coarse-grained approach the change of GPGPU produces a completely different pattern on the execution time. While using the C1060 the model does not depend on on the number of attributes, while using the C2070, the execution times get lower and also vary logarithmically depending on the number of attributes.

⁴A random sampling was used to shorten the training set size while oversampling was used to produce the opposite effect.

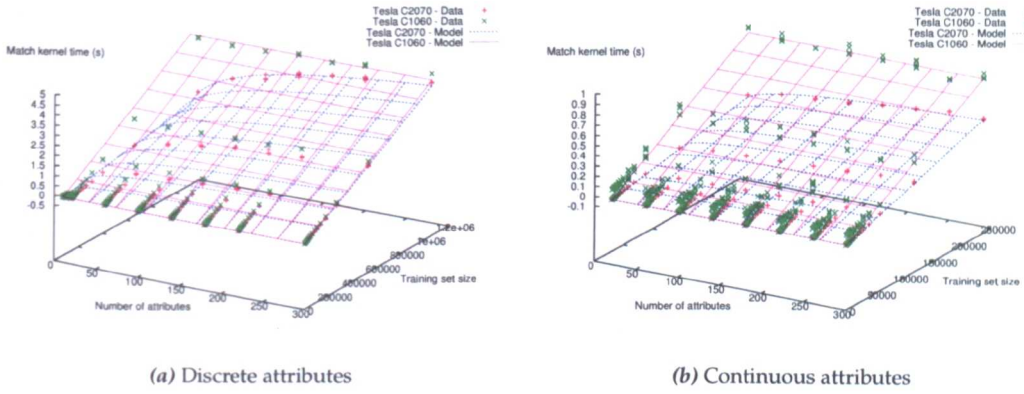


Figure 8.12: Execution time model of the match kernel for different types of attributes

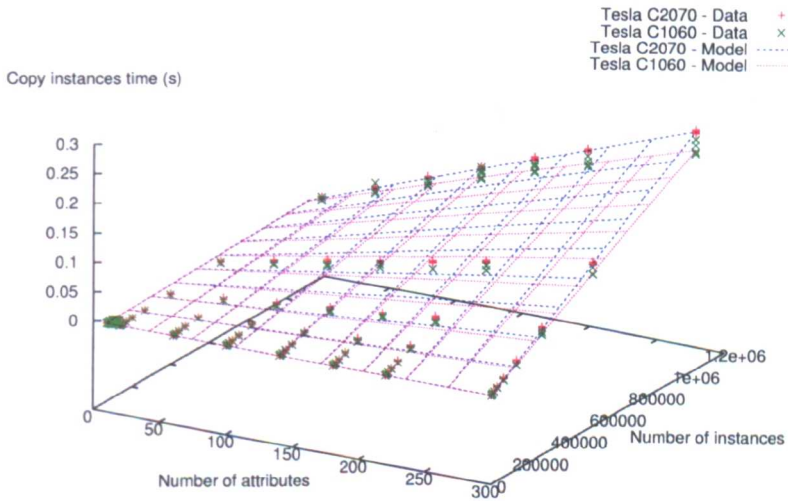


Figure 8.13: Execution time model of the instance copy operations

Regarding the execution time of the instances' copy operations (Figure 8.13) the models are very similar even when changing the GPGPU. Particularly in this case the model grows linearly with the number of attributes and the number of instances. However, it seems like the C1060 card has slightly better memory bandwidth than the C2070.

Having an approximation of how much the match and the copy operation process take in the device we can try to approximate how much will it take a simple GA evaluation with i iterations. Assuming the simplest case, in which all iterations handle exactly the same number of windows w the time taken by the evaluation process is:

$$eval(att, ins, i, w) = copyins(att, ins) + i \cdot match\left(att, \frac{ins}{w}\right) \quad (8.7.1)$$

Figures 8.14 and 8.15 show the validation of the previous model against the data shown in Section 8.5 for the continuous and discrete problems. Small problems are not shown as both models underestimate these cases. In these figures we can observe that the model behaves as a lower bound in most cases, which is expected since we only considered the operations that took

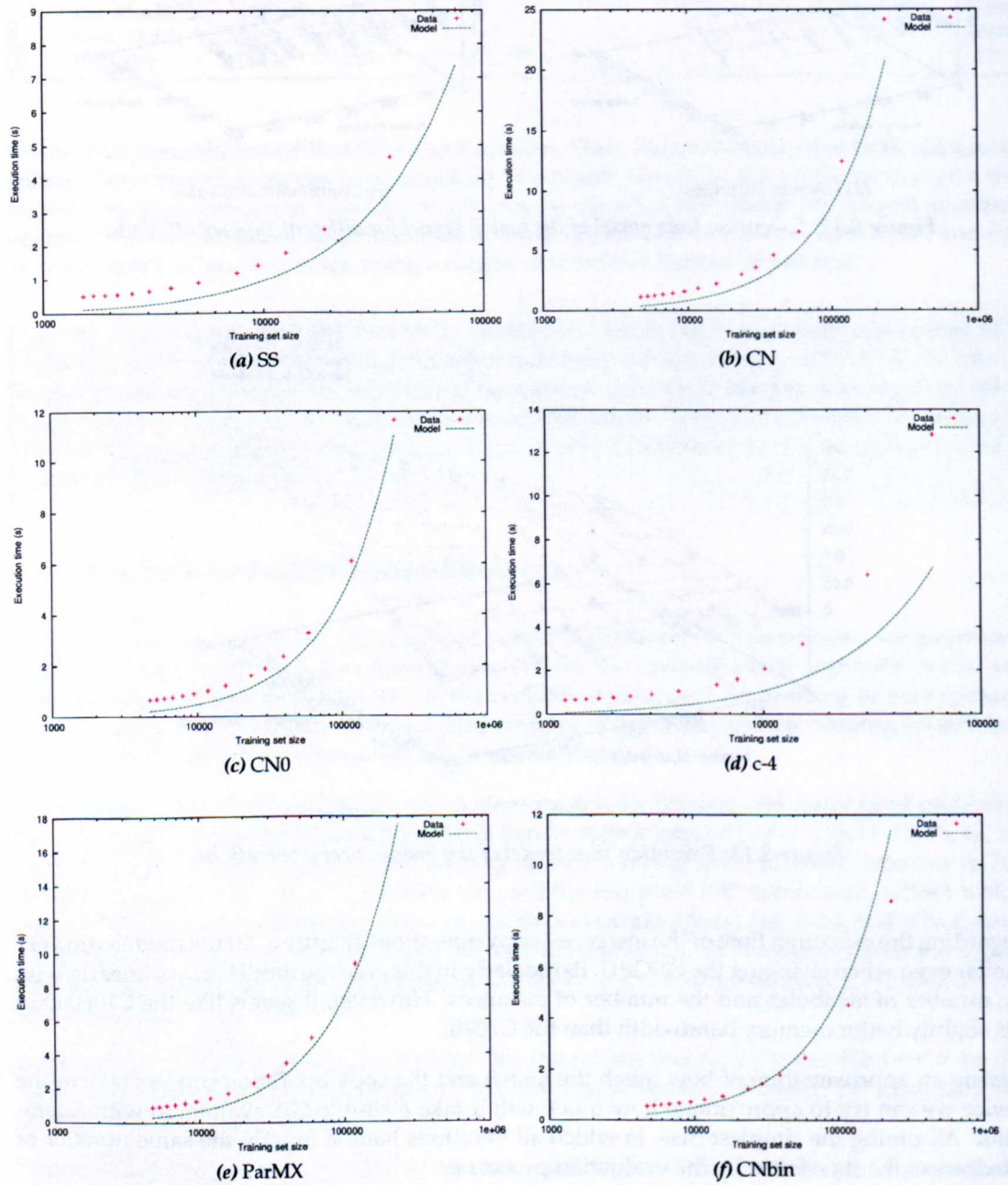


Figure 8.14: Model validation over the data obtained in the previous sections for the coarse-grained approach over the Tesla C2070

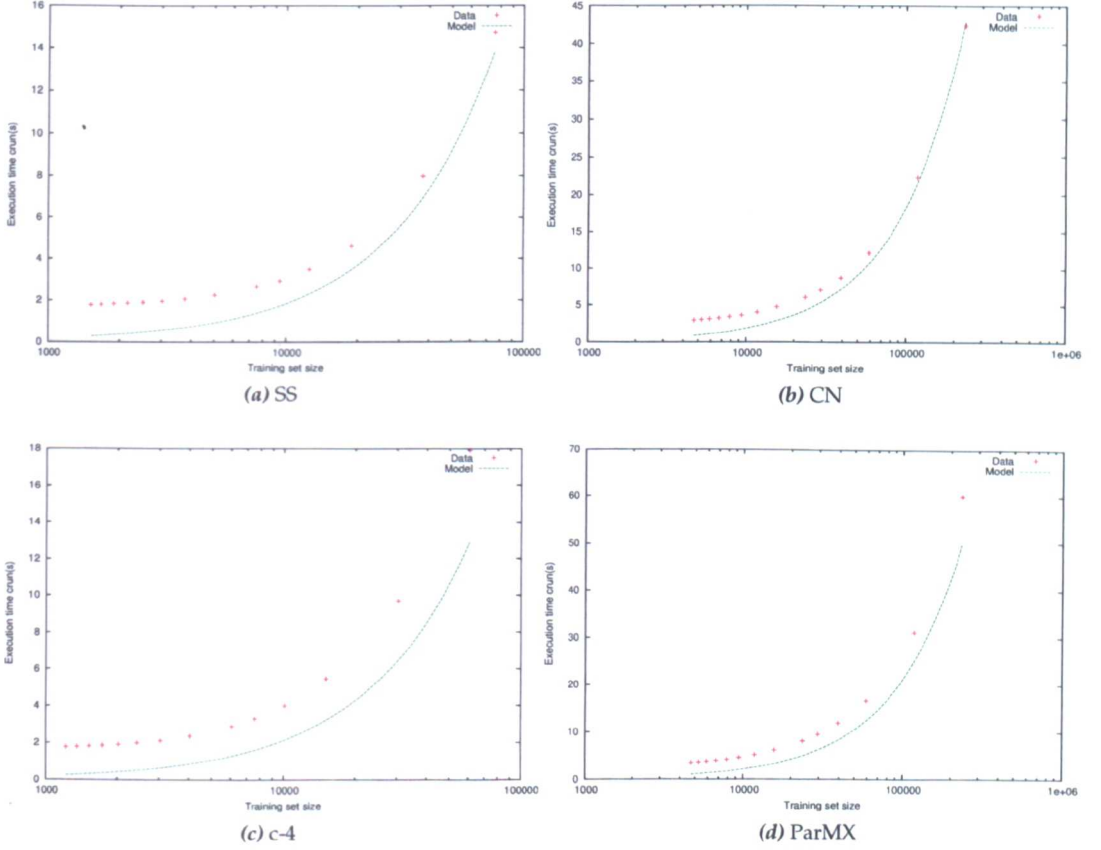


Figure 8.15: Model validation over the data obtained in the previous sections for the coarse-grained approach over the Tesla C1060

most of the time. For the model on the C2070 there are only two problems for which the model does not fit the data accordingly: *c-4* and *CNbin*. In the case of *CNbin* the problem with the fitting might be because this problem has very little attributes and it is right at the border of the sample space used to create the models. In the case of the model for the C1060 (Figure 8.15) we can observe that since this model does not have an independent term there is a subestimation when the training set gets very small.

Regarding the mixed attributes the most intuitive way to generate a model is assuming that if the problem has n discrete attributes and m continuous attributes the evaluation time will be equal to:

$$eval(n, m, ins, i, w) = copyins \left(n + m, \frac{ins}{w} \right) + i \cdot \left(mismatch(n, ins) + cmatch \left(m, \frac{ins}{w} \right) \right) \quad (8.7.2)$$

Figures 8.16 and 8.17 show the validation of this model over mixed problem on the C2070 and the C1060, respectively. In these figures we can observe that generating a model for discrete and continuous attributes independently is not sufficient to model the behaviour of the system on mixed domains. In these domains the way the different types of attributes are interleaved might affect the execution times. Moreover, the fact that a change in the GPGPU changes the model introduces the idea of generating a model (if possible) that takes in consideration intrinsic characteristics of the device, in order to understand better these behavioural changes.

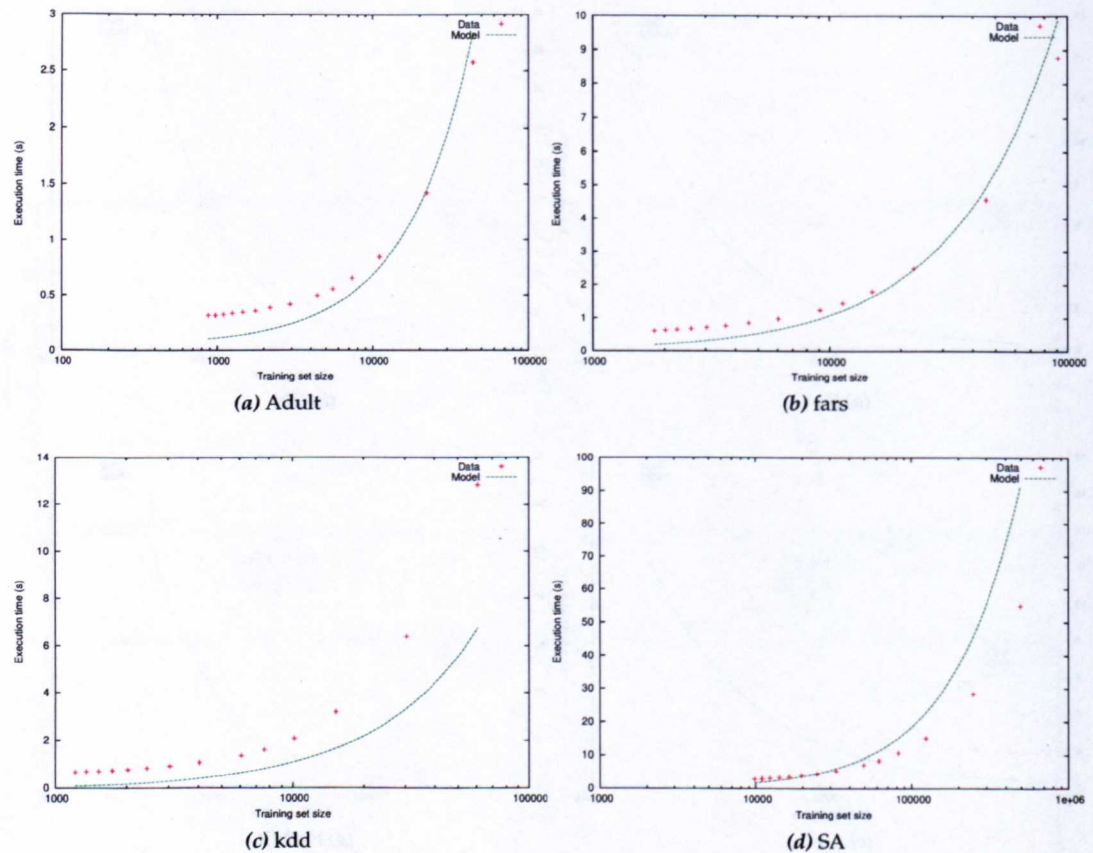


Figure 8.16: Model validation over the mixed problems for the coarse-grained approach over the Tesla C2070

8.8 Conclusions

We have successfully implemented a GPGPU-based evaluation process for the BioHEL system. Two different parallelisation paradigms were analysed: the *coarse-grained paradigm* which is more conservative and only parallelises the evaluation of instances and classifiers, and the *fine-grained paradigm* which expands the parallelism of the previous one to the attribute dimension. Even though it seems logical that the larger the parallelism the larger the speedup, this is not entirely true, since the number of multiprocessors in the GPGPU is finite, and if we assign a number of tasks way above its capacity they are going to be serialised.

In this work we have determined that the right balance between parallelism and thread workload produces the best results. In this sense the coarse-grained approach is the one that gives the best results. This paradigm (when run on a Tesla C1060) achieves a maximum speedup of 52.4X (when evaluated on its own) and up to 58.1X when integrated within BioHEL. Even though these values are dependant on the characteristics of the dataset and the architecture used, we have shown that in general the CUDA architecture can be used to successfully speed up the evaluation process of BioHEL when handling large problems. Moreover, the combination of the CUDA-based evaluation and the ILAS windowing scheme also showed to be beneficial, obtaining a maximum combined speedup of 765.3X on a Tesla C1060.

This implementation exploits the intrinsic parallelism in the evaluation process and can be easily extended to other EL paradigms. The speedup obtained with CUDA in the evaluation

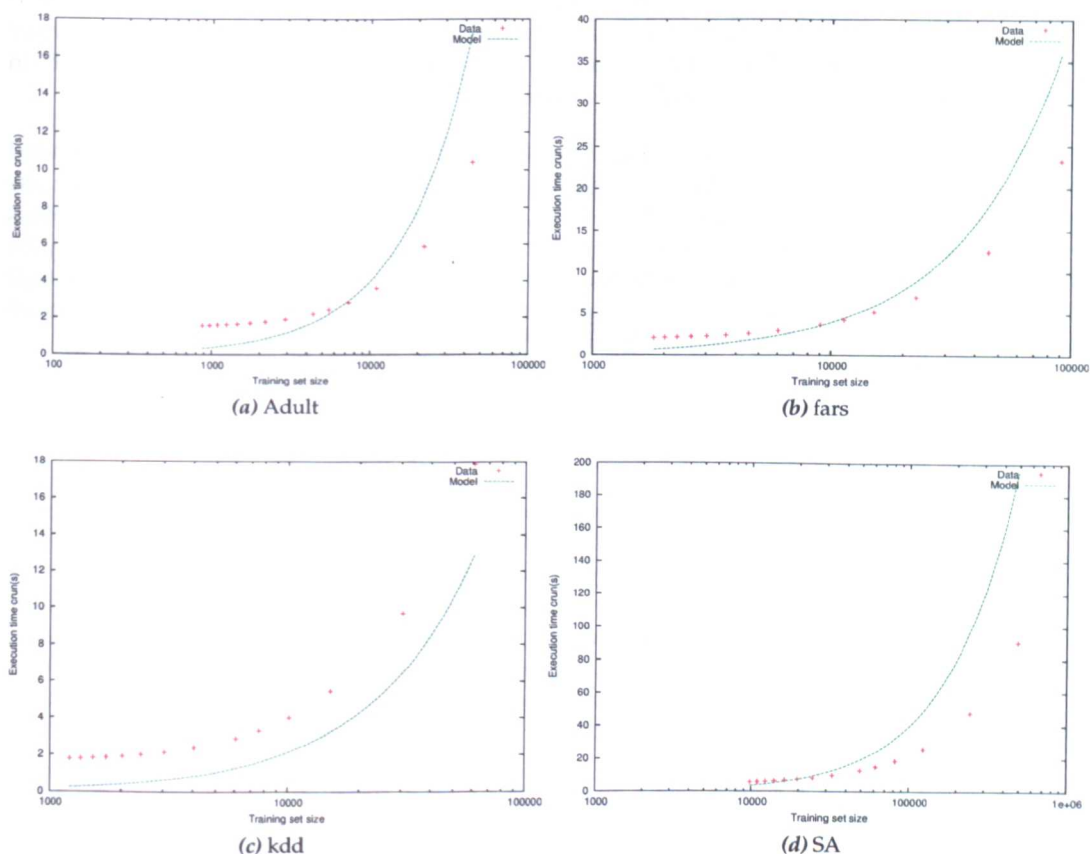


Figure 8.17: Model validation over the mixed problems for the coarse-grained approach over the Tesla C1060

process helps the system to handle larger problems faster, one of the limitations of the system found on Chapter 4.

Also models were generated to explain the behaviour of the execution time according to the type of problem, number of attributes and training set size. In this sense the models manage to explain the behaviour for most problems. However, when the GPGPU changes the model structure changes entirely which questions the benefits of having these types of models. In this sense we conclude it is best to generate more complex models that include intrinsic characteristics of the GPGPU to explain the algorithm's behaviour.

8.9 Further work

As further work we would like to extend this implementation so it can use more than one GPGPU at the time. By doing this, the system could schedule the matching of different groups of classifiers and instances in different GPGPUs, if more than one device is available.

It could be also interesting to develop an implementation that copies the information only once into device memory and synchronises the population and the instances when there are changes. This way, the system would only perform small memory copy operations which would be interesting to compare with the current system.

Finally, the use of GPGPUs opens the door to perform much extensive experiments that we could not afford to perform before. For instance, it is known that increasing the number of strata of the ILAS windowing scheme makes the problem more difficult to learn [Bacardit et al., 2004].

Nevertheless, it was not possible to exhaustively determine when this was producing a significant impact in the system's performance in large scale domains because the experiments were too computationally demanding. GPGPUs allow us to perform much larger experiments than before, thus we are able to push forward the boundaries of evolutionary data mining.

In general, the parallelisation methodologies presented in this chapter can be easily extended to speed up the match or the evaluation process in other EL systems. For systems which use a supervised learning approach (Pittsburgh-like LCS, IRL, etc) the evaluation process involves the match process and the fitness calculation of the classifiers while for Michigan LCS it only involves the creation of the *match set*. While the first ones could potentially have more advantage of this parallelisation techniques, the later could apply the techniques explained in this thesis to match the examples that arrive through online learning.

Post-processing the final rule sets

One last weakness to tackle in BioHEL is the length and generality of the solutions generated by means of the IRL paradigm. Towards this aim this chapter introduces three post-processing operators (rule cleaning, rule pruning and rule swapping) which combined together in different ways can help reduce the complexity of the rule sets generated with BioHEL. The first two operators edit individual rules to reduce their number of expressed attributes and hence make them more general. The latter operator changes the order of the rules within the decision list (based on the similarities between them) to identify and delete the unnecessary ones. The results show that it is possible to both reduce the number of specified attributes per rule and the number rules by up to 30% in some problems without producing significant accuracy changes.

9.1 Introduction

A frequent theme throughout the years in EL research is the proposal of methods to reduce the complexity and/or improve the generality of the generated solutions (mostly rule sets) to provide a more understandable solution for the user. After the learning process of all kinds of EL systems, the generated rule sets may not be optimal. There are many reasons for this: a) In IRL the greedy nature of the learning process makes it easy to generate suboptimal rule sets, b) the Michigan approach usually generates very large rule sets of fixed size where some of the rules are duplicated or irrelevant, and c) in all paradigms there is no guarantee that the solutions are as optimal as possible (minimal rule sets). These problems get aggravated while the problems become larger and complex, as the final solutions increase their complexity as well. Therefore, it is important to edit the rules and rule sets to make them more general, without compromising their prediction capacity. Towards this aim, some studies in the literature focus their efforts in reducing the cardinality of the final rule sets [Dixon et al., 2003; Wilson, 2001a]. Other studies present intelligent operators to combine the rules in order to obtain less complex solutions [Bacardit and Krasnogor, 2009b; Butz, 2006].

Throughout Chapter 4 we noticed that the rule sets generated with BioHEL present two problems. First, the rule sets generated by BioHEL, even though they are small compared to the ones obtained with Michigan LCS, they might not be maximally general since they are generated in a greedy fashion (without global supervision) and they are considerably larger than the ones obtained with GAssist. By doing this the rules might be learned in the wrong order, which causes the solution to be larger than the optimal solution. The second problem is that the individuals in the initial populations in BioHEL are generated randomly with a user-defined specificity (number of relevant attributes and instance space covered by these attributes, which in binary domains are defined by the parameters *ExpAtts* and *p*, respectively) and the learning process is in charge of evolving these rules and adapt them to the correct specificity of the problem (which for binary domains is equal to the *k* of the problem). Even though this is accomplished in most cases by means of the coverage breakpoint, it can happen that the good rules are more

specific than what they should be and they need to be simplified, as shown in Chapter 7.

To overcome these two problems, this chapter proposes three local search operators (*rule cleaning*, *rule pruning* and *rule swapping*) which combined together in different ways can help reduce the cardinality of the generated solutions as well as the number of attributes expressed in each rule. While the first two act on individual rules to reduce the number of attributes expressed without degrading the accuracy, the last one is a simple heuristic that changes the order of the rules considering similarities (or overlapping) between them.

These rule editing operators are designed for the GABIL and hyper-rectangle knowledge representations, which encode discrete and continuous attributes, respectively (See Section 2.4.2.1). Even though these operators are evaluated in this chapter over rule sets obtained with BioHEL, they can be applied directly to other systems employing decision lists and the GABIL or hyper-rectangle representations (e.g. GAssist, GALE, NAX) and can easily be adapted to many other rule representations.

In the first stage of experiments each operator is evaluated independently. Afterwards, combinations between the operators were tested to determine if applying them together in a particular order improves the results. The improvements were measured based on the number of attributes, number of rules and accuracy observed before and after applying the operators.

The results show that these operators can improve the final rule sets significantly without degrading the test accuracy. It is shown that when combining the operators in particular ways it is possible to reduce both the number of attributes and rules in the initial solution by up to 30%. However, the reduction percentage varies among problems.

This chapter is organised as follows. Section 9.2 describes previous works that apply operators or mechanisms to improve either individual rules or complete rule sets. Section 9.3 explains in detail the proposed post-processing operators. Section 9.4 presents the experimental design to evaluate these operators. Section 9.5 presents the results obtained after evaluating the operators and Section 9.6 presents the conclusions and future research directions.

9.2 Related work: Rule optimisation algorithms

The operators presented in this chapter are local search operators that fall into the category of Memetic Algorithms (MA) [Krasnogor and Smith, 2005]. However, instead of being applied during the learning process, these operators are used to post-processes and compact the final solution. The Memetic Pittsburgh LCS (MPLCS) [Bacardit and Krasnogor, 2009b] proposed several operators with some similarities to the operators presented in this thesis. This work presents four operators, three rule-wise operators and one rule-set-wise operator. Similarly to the operators presented in this chapter, they work with the GABIL representation. The first rule-wise operator *Rule Cleaning* disables the literal in the GABIL string that makes the rule cover the most misclassified examples and does not help to classify correctly any example at all. The *Rule Generalizing* operator has the opposite effect, it activates the literal that correctly classifies more examples and does not increase the misclassifications. The *Rule Splitting* operator separates the rule into two rules, over which the cleaning operator can be applied more effectively. Moreover, this work also presents a rule-set-wise operator in which a new rule set is generated by inserting the rules of parent rule sets in the order that increases the most the accuracy of the offspring rule set.

The similarity between the rule cleaning mechanism presented in [Bacardit and Krasnogor, 2009b] and our work is given by the fact that both operators disable literals in the rule that contribute to misclassify examples. The difference is that our cleaning operator does not attempt to make one change in the rule, but to make all the necessary changes needed (according to a predefined policy) in each one of the attributes. There is a similarity as well, between the

rule generalising and the rule pruning mechanism presented in this chapter. While the rule generalising operator tries to activate the bit that makes the classifier cover more examples without compromising the accuracy, our pruning mechanism eliminates all the attributes from the rule that can be deleted without compromising accuracy or covering negative examples. In this sense we could say that our operators are more aggressive than the previous ones, but this is needed since we are acting over final rule sets and we need to make all the necessary changes in a single application of the operator. Moreover, we do not need to worry about the balance between exploration and exploitation which is crucial for MPLCS. Finally, MPLCS's rule-set-wise operator although also designed for decision lists is quite different from the rule swapping presented in this chapter. The MPLCS operator creates a new rule set from scratch from the rules of many parents, while our operator edits a single rule set swapping the order of similar rules.

Other approaches of rule optimisation use Estimation of Distribution Algorithms (EDA) [Laragaña and Lozano, 2002] to model the structure of the problem and consider this knowledge to generate new individuals. Examples of this are the crossover operators implemented by Butz [Butz, 2006] that use either an ECGA [Harik, 1999] or a BOA [Pelikan et al., 1999] to determine the structure of the building blocks of the problem. Also the Compact Classifier System (CCS) [Llorà et al., 2005] and its extension [Llorà et al., 2006] uses EDAs within Pittsburgh LCS to generate new individuals based on the generated model.

Regarding the compaction of the final rule sets a lot of work was done over Michigan LCS. The first attempt to reduce the cardinality of the rule sets was presented in [Kovacs, 1997; Wilson, 1995], as a population *condensation phase* where mutation and crossover were disabled. Afterwards, the Compact Ruleset Algorithm (CRA) [Wilson, 2001a] was proposed, which orders the rules from best to worst according to a certain characteristic (e.g. experience or numerosity), and keeps only the necessary amount of rules that produces the same accuracy as before. This algorithm reduces dramatically the number of rules but it also has very high complexity and does not work well if the rule set has not reach maximum accuracy. To tackle the complexity problem Dixon et al. [2003] presented an algorithm based on marking the "useful" rules. The useful rules are the rules that after testing the whole training set, they fell into the action set at least once. Two possible approaches can be followed: marking the only the rule with the highest prediction accuracy in the action set, or marking all the rules in the action set. This approach produces similar results in much less computational time. Moreover, Orriols-Puig and Bernadó-Mansilla [2004] presents a detailed analysis of the three reduction algorithms mentioned before, where they were compared using paired t-tests. In this analysis it was shown that Wilson's algorithm produces the best compaction with less degradation, while the Dixon approaches are more efficient in terms of computational time but produce more accuracy degradation.

Other approaches of rule set compaction can be found in [Fu and Davis, 2002; Gao et al., 2006]. Fu and Davis [2002] present two modifications of the CRA specifically designed to handle not well-trained rule sets (solutions that have not achieved maximum accuracy). The first modification was the deletion of classifiers incrementally. While the original algorithm works at a macro-classifier level, this algorithm erases the classifiers one by one. The second modification optimises the construction of the final set of rules, since this is the stage that takes more computational time. Moreover, Gao et al. [2006] also presents a version of the original CRA algorithm that instead of trying to find the useful subset it erases classifiers one by one, reducing the computational complexity of the algorithm. Statistical test were not performed in any of these works to determine the differences with the previous approaches.

It is worth mention that in all these algorithms the rule set reduction is done at the cost of reducing the test accuracy slightly. Also these methods can not be directly compared to the one presented in this chapter because they are specifically designed for Michigan LCS, where an order between the rules does not exist, in contrast to decision lists.

Problem: $x1 = 1 \wedge x3 = 0$

000 = 0 100 = 1

001 = 0 101 = 0

010 = 0 110 = 1

011 = 0 111 = 0

Good rule

$x1 = 1 \wedge x3 = 0$

Over-specific rules

$x1 = 1 \wedge x2 = 1 \wedge x3 = 0$

$x1 = 1 \wedge x2 = 0 \wedge x3 = 0$

Figure 9.1: Example of good rules and over-specific rules for a particular problem with 3 attributes.

9.3 Post-processing operators for BioHEL

In this section we discuss three different rule set post-processing operators: a) rule pruning, b) rule cleaning and c) rule swapping. The first two operators act on individual rules while the last one is a rule set compaction operator. The next section explains each one of them in greater detail.

9.3.1 Rule Pruning

The rule pruning operator acts by dropping attributes from the rule, making them more general. Most learning systems introduce generality pressure in their functioning (by means of their fitness function/subsumption mechanisms, etc.). In particular for the BioHEL system, this generality pressure is applied by means of the coverage term (controlled by the coverage breakpoint parameter) in the fitness function. Hence, at the end of the learning process the generated rules are expected to be maximally general and specify only relevant attributes that, if made irrelevant, will decrease the accuracy of the rule. However, this does not always happen. This operator tackles the scenarios where the maximal generality of the rules has not been reached.

Figure 9.1 presents simple examples of over-specific rules for a small problem of 3 attributes. As we can see having over-specific rules, not only makes the solution more complex, but also makes the solution larger since each rule covers a smaller percentage of the training set.

This operator (identified as PR) works on individual rules iterating along each one of their attributes. The system tries to erase each attribute, one at the time, and recalculates the accuracy. If the accuracy decreases (because the rule mismatches more instances than before) the change is reverted. The pseudocode for this operator can be found in Algorithm 9.3.1. This operator is the same as the one applied in Chapter 7 to prune the candidate representatives found in an initial population.

Algorithm 9.3.1: PRUNING(Classifier $c1$)

```

prevacc ← GETACCURACY( $c1$ )
for each att ∈ GETATTRIBUTES( $c1$ )
do {
  REMOVEATTRIBUTE(att,  $c1$ )
  if GETACCURACY( $c1$ ) ≥ prevacc
  then prevacc ← GETACCURACY( $c1$ )
  else RESTORE(att,  $c1$ )
return ( $c1$ )

```

This operator is very simple and does not depend on the representation used. It can be easily translated to other representations that do not use attribute lists, by just generalising attributes

Algorithm 9.3.2: CLEANRULE(*c*)

```

matchExamples ← GETMATCHEDEXAMPLES(c)
for i ∈ c.atts
  if i is continuous
    then { sortedPositive = matchExamples.positive[i].sort()
           c[i].lowerBound ← sortedPositive[0]
           c[i].upperBound ← sortedPositive[sortedPositive.length - 1] }
    else { for pred ∈ i.values
           do { pos ← i.values.index[pred]
                if pred == 1 ∧ matchExamples.positive[i][pos] == ∅
                then pred ← 0 }
    return (c)

```

To “clean” rules, the system calculates the instances classified by this rule. Afterwards, it iterates along the attributes reducing the covered space to match preferably only the positive examples (examples correctly classified). In the hyper-rectangle representation this involves reducing the distance between the lower and upper bounds of an attribute’s interval, and in the GABIL representation it involves disabling (setting to 0) some of the values of an attribute’s predicate. Two different policies (shown in Figure 9.3) can be followed to shrink the domain of an attribute:

CL - Focus on the positive examples. This policy reduces the predicate to cover only the positive values observed. In the case of continuous attributes this policy places the boundaries right next to the first and last positive value observed. In the case of discrete attributes it leaves activated only the literals that correspond to the values in the positive examples. Algorithm 9.3.2 shows the pseudocode for this operator.

Algorithm 9.3.3: CLEANRULE2(*c*)

```

matchExamples ← GETMATCHEDEXAMPLES(c)
for i ∈ c.atts
  if i is continuous
    then { sortedPos = matchExamples.positive[i].sort()
           sortedNeg = matchExamples.negative[i].sort()
           lowNeg ← CLOSESTNEGLEFT(matchExamples.negative[i,
                                sortedPos[0])
           upNeg ← CLOSESTNEGRIGHT(matchExamples.negative[i,
                                sortedPos[sortedPos.length - 1])
           c[i].lowerBound ←  $\frac{\text{sortedPos}[0] + \text{lowNeg}}{2}$ 
           c[i].upperBound ←  $\frac{\text{sortedPos}[\text{sortedPos.length} - 1] + \text{upNeg}}{2}$  }
    else { for pred ∈ i.values
           do { pos ← i.values.index[pred]
                if pred == 1 ∧ matchExamples.negative[i][pos] ≠ ∅ ∧
                   matchExamples.positive[i][pos] == ∅
                then pred ← 0 }

```

CL2 - Do not infer. This policy is more conservative than the previous one. In the case of the continuous attributes, it places the boundaries between the first positive value observed and the closes negative value. In the case of discrete attributes this policy only deactivates the literals that appear in the negative examples and do not appear in the positive ones. In order words, this policy tries not to infer what happens in the areas of the search space where there is no input. Algorithm 9.3.4 shows the pseudocode for this operator.

Algorithm 9.3.4: CLEANRULE2(c)

```

matchExamples ← GETMATCHEDEXAMPLES(c)
for i ∈ c.atts
  if i is continuous
    then {
      sortedPos = matchExamples.positive[i].sort()
      sortedNeg = matchExamples.negative[i].sort()
      lowNeg ← CLOSESTNEGLEFT(matchExamples.negative[i],
                              sortedPos[0])
      upNeg ← CLOSESTNEGRIGHT(matchExamples.negative[i],
                              sortedPos[sortedPos.length - 1])
      c[i].lowerBound ←  $\frac{\text{sortedPos}[0] + \text{lowNeg}}{2}$ 
      c[i].upperBound ←  $\frac{\text{sortedPos}[\text{sortedPos.length} - 1] + \text{upNeg}}{2}$ 
    }
  else {
    for pred ∈ i.values
      do {
        pos ← i.values.index[pred]
        if pred == 1 ∧ matchExamples.negative[i][pos] ≠ ∅ ∧
           matchExamples.positive[i][pos] == ∅
        then pred ← 0
      }
  }

```

9.3.3 Rule Swapping

Rule swapping (SW) is a rule set modifier operator which main goal is to reduce the number of rules in a decision list by changing their order. Particularly in IRL the rules are generated one after another without having a global supervision of the generated rule sets. This could lead to the generation of larger rule sets than necessary. Moreover, it could be the case that the system makes mistakes while learning the first rules, and then more rules are necessary to cover the rest of the examples left in the training set. An example of this situation is shown graphically in Figure 9.4 where if Rule A is learned before Rule B, it is necessary to learn extra rules to cover the rest of the examples. This does not happen if the rules are discovered in the inverse order. Therefore, if we change the order of the rules in the final rule sets we could potentially fix some of the errors the system made during the learning process and reduce the number of classifiers of the final solution.

This simple heuristic iterates (from top to bottom) through the rules as shown in Algorithm 9.3.5. In each iteration the rule being evaluated is compared to all rules placed after it within the decision list, in order to find the most “similar” rule to the observed, and swaps them. Afterwards, the accuracy of the rule set is recalculated (considering the new order) and if the accuracy decreases the change is reverted. If the change is approved, the system scans all the

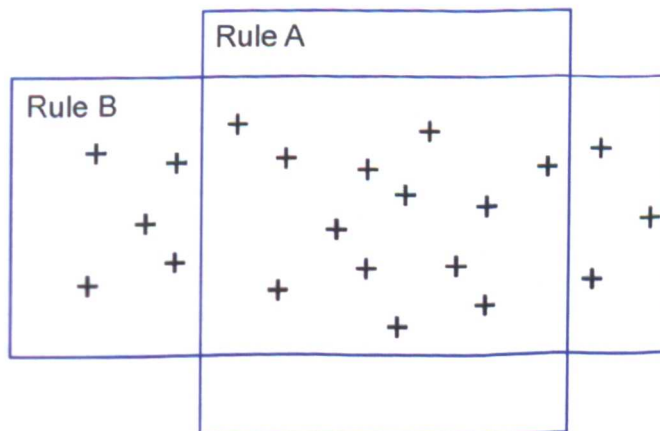


Figure 9.4: Example case where rule swapping can be beneficial since it allows erasing extra rules

rules that follow the first swapped rule, erasing the ones that are not active anymore (rules that do not cover any example) because they have been subsumed by the change in order.

Algorithm 9.3.5: RULESWAPPING(*rs*)

```

accuracy ← ACCURACY(rs)
for r ∈ rs
    rs2 ← FOLLOWINGRULES(r, rs) // Returns the set of rules placed after r in rs
    r1 ← MOSTSIMILARRULE(r, rs2)
    SWAP(r1, r, rs)
    newAcc ← ACCURACY(rs)
    if newAcc < accuracy
        then SWAP(r, r1, rs)
    do { else if newAcc > accuracy
        then accuracy ← newAcc
        rs2 ← FOLLOWINGRULES(r1, rs)
        for f ∈ rs2
            do { if f.coverage == 0
                then DELETE(f, rs) // Deletes rules with no coverage
    }
```

The similarity metric between rules depends on the percentage of the space the rules match in common, and its calculation is different for discrete and continuous attributes. Considering rules *i* and *j*, both containing a continuous attribute *c*, the area this attribute matches in common for both rules (intersection) is:

$$S_c(i, j) = \frac{\min(ub_c(i), ub_c(j)) - \max(lb_c(i), lb_c(j))}{\max(ub_c(i), ub_c(j)) - \min(lb_c(i), lb_c(j))}$$

where $lb_c(i)$ and $ub_c(i)$ can be read as lower and upper bound of attribute *c* in rule *i*. For the discrete attributes and GABIL representation the intersection between attributes is calculated as $S_c(i, j) = coincidentVals_c(i, j)$, where $coincidentVals_c(i, j)$ is the number of values that are activated in both predicates at the same time.

Moreover, to calculate the similarities using the ALKR representation we have to take into account that some attributes may not appear in the rules. Therefore, we have to consider three cases: a) the attribute appears in both rules, b) the attribute appears only in one of the rules, and c) the attributes does not appear in any of the rules. If the attribute appears in only one of the predicates the intersection is done considering the second attribute covers the whole domain of the attribute. If the attribute does not appear in any of the predicates a total coincidence is considered. Therefore, the similarities between rule *i* and rule *j* can be calculated as follows:

$$S(i, j) = \frac{Dis}{NA} \frac{\sum_k^{Dis} S_k(i, j)}{\sum_k^{Dis} numVals(k)} + \frac{Real}{NA} \sum_k^{Real} S_k(i, j) + \frac{Mi}{NA}$$

where *Dis* is the number of discrete attributes observed in both rules, *Real* is the number of real attributes observed, *Mi* is the number of attributes that do not appear in any of the predicates (missing attributes), *NA* is the total number of attributes in the problem and $numVals(k)$ is the number of values of attribute *k*. This formula sums the similarities for the discrete and continuous attributes independently, and then weights these values according to the number of real or discrete attributes observed. Also, at the end, this formula considers a complete intersection between all the attributes that do not appear in any of the lists. It is worth noticing that we normalise the discrete attribute term according to the number of values observed after summing the values for all attributes, to treat all attributes equally in problems with discrete attributes of varying cardinality.

9.4 Experimental design

The experiments in this chapter are divided in three parts. The first part shows the impact of each operator independently. The two other parts show combinations of them (in terms of the order of application) to better understand how these mechanisms interact between each other.

We tested our post-processing operators (CL and CL2 -rule cleaning with both policies, PR -rule pruning and SW -rule swap) over rule sets generated using BioHEL in the experiments reported in Chapter 4. Specifically, we used the rule sets generated with BioHEL using the standard configuration shown in Table 4.2 plus three parameters that vary depending on the size of the problem according to Chapter 4: the coverage breakpoint, the number of ILAS windows and the default class policy. For large problems (over 40000 instances) a coverage breakpoint of 0.001, 10 windows and majority default class was used. For small problems (less than 40000 instances) a coverage breakpoint of 0.1, 2 windows and disabled default class was used. Moreover, in Chapter 4 stratified ten fold cross-validation was employed, and five rule sets were generated from each training set. Thus, for each problem we tested our operators on 50 (potentially different) rule sets. Furthermore, our operators are not stochastic, so we run these only once per rule set.

Our results are presented in terms of the percentage of variance (with respect to the initial results) in test accuracy, number of rules and number of attributes. We also show the test accuracy obtained after integrating the rule sets into a simple consensus ensemble (see Section 2.4.1.4). We have included the ensemble accuracy because we were interested in determining if these operators increase or reduce the diversity of the ensemble which is the key to its success. For the comparison with the initial rule sets, we present their results in Table 9.1. Moreover, we show the execution times of applying the different operators over large problems (experiments were run in Intel Xeon E5472 3.0GHz processors) to determine how they scale in these domains. At the end of our analysis, we determine if there are significant differences between the results obtained with the different operators, the combinations of them and the initial results all together using the Friedman test and the post-hoc Holm test.

9.5 Results

In this section we will first analyse the impact of the operators independently. Later, we analyse the impact of combining the rule operators CL and PR. Afterwards, we analyse the combination with the rule set operator SW. Finally, we analyse statistically the results obtained with all the combinations of operators shown.

9.5.1 Independent operator analysis

Figure 9.5 shows the results of the first stage of experiments. We can observe that only a small fraction of problems present large accuracy variations. Interestingly, the variations are larger (both in terms of accuracy increase and decrease) when looking at the ensemble accuracy than when looking at the average test accuracy, and this is a trend that will be observed also in later stages of experiments. Moreover, we can see that the PR operator can reduce the number of expressed attributes up to 23% in large problems and up to 13% in small problems. Even though the number of attributes expressed per rule in BioHEL is not very high, this result shows that is it possible to generate even more general solutions without varying the test accuracy. On the other hand, SW could potentially reduce the number of attributes as a consequence of deleting rules, but its impact over the attributes is not very big, it is only observed in the *CN-bin* problem. Moreover, we can also observe that the SW operator is capable of erasing up to 30%

Table 9.1: Test accuracy, test accuracy after ensembles, number of rules and number of attributes of the rule sets before post-processing

Prob.	Test acc	Ensemble acc	Rules	Atts
Adult	85.19 \pm 0.55	86.09 \pm 0.39	194.24 \pm 10.26	10.18 \pm 2.80
C-4	79.20 \pm 0.50	80.94 \pm 0.46	316.14 \pm 19.10	9.96 \pm 3.23
KDDCup	99.93 \pm 0.02	99.95 \pm 0.01	188.84 \pm 13.52	4.25 \pm 2.99
ParMX	99.99 \pm 0.01	100.00 \pm 0.00	394.34 \pm 19.39	9.00 \pm 0.01
SS1	65.20 \pm 1.02	71.20 \pm 1.06	773.26 \pm 30.42	11.49 \pm 3.40
CN	68.68 \pm 0.69	80.59 \pm 0.46	253.34 \pm 12.48	10.09 \pm 2.78
CN-bin	72.45 \pm 0.47	72.44 \pm 0.49	38.20 \pm 1.85	7.12 \pm 0.73
bal	85.34 \pm 4.55	86.55 \pm 3.64	30.34 \pm 2.03	3.09 \pm 0.49
bpa	62.67 \pm 7.28	67.87 \pm 5.59	20.18 \pm 1.12	3.35 \pm 1.29
bre	66.69 \pm 8.22	68.56 \pm 8.51	22.50 \pm 1.36	3.64 \pm 1.52
cmc	53.30 \pm 4.50	54.18 \pm 4.64	14.28 \pm 5.71	2.58 \pm 1.39
col	95.07 \pm 3.07	97.32 \pm 2.15	6.38 \pm 0.78	2.40 \pm 1.37
cr-a	83.37 \pm 5.42	84.90 \pm 5.49	20.22 \pm 1.57	3.98 \pm 1.71
gl	70.98 \pm 10.51	75.64 \pm 9.22	17.14 \pm 1.07	3.18 \pm 1.50
h-cl	78.07 \pm 6.54	81.23 \pm 4.95	13.16 \pm 1.11	3.96 \pm 1.46
hep	87.40 \pm 8.30	89.79 \pm 8.10	5.92 \pm 0.70	2.72 \pm 1.03
h-h	94.97 \pm 3.58	95.92 \pm 3.52	5.76 \pm 0.89	2.20 \pm 1.29
h-s	76.44 \pm 8.56	78.89 \pm 8.20	14.24 \pm 1.13	4.04 \pm 1.55
ion	89.46 \pm 3.57	92.27 \pm 2.81	5.04 \pm 0.83	4.74 \pm 1.88
irs	93.87 \pm 5.01	94.00 \pm 4.92	6.32 \pm 1.02	1.57 \pm 0.56
lab	97.14 \pm 5.77	97.14 \pm 6.02	2.00 \pm 0.00	1.60 \pm 0.49
lym	79.37 \pm 13.57	80.89 \pm 13.09	9.52 \pm 0.99	3.01 \pm 1.29
pen	92.31 \pm 1.98	94.62 \pm 1.04	27.82 \pm 2.80	6.46 \pm 3.44
pim	71.19 \pm 5.11	73.97 \pm 4.44	24.56 \pm 1.43	4.23 \pm 1.60
prt	45.95 \pm 8.63	47.56 \pm 8.99	28.32 \pm 4.43	4.34 \pm 1.99
sat	85.13 \pm 1.52	86.99 \pm 1.39	31.02 \pm 2.32	5.68 \pm 3.57
son	73.85 \pm 10.28	80.56 \pm 12.40	7.28 \pm 0.67	3.72 \pm 1.51
thy	93.20 \pm 4.14	93.48 \pm 3.93	7.16 \pm 0.87	1.69 \pm 0.73
vot	96.09 \pm 3.42	96.10 \pm 3.79	13.66 \pm 2.22	2.22 \pm 0.97
wav	79.67 \pm 1.85	83.10 \pm 1.75	33.40 \pm 1.65	6.40 \pm 2.55
wbcd	94.85 \pm 2.33	95.57 \pm 2.17	6.70 \pm 1.43	3.66 \pm 1.94
wdbc	94.53 \pm 2.99	95.64 \pm 3.35	7.60 \pm 0.83	2.84 \pm 1.13
wine	91.75 \pm 6.26	95.39 \pm 4.52	3.58 \pm 0.54	2.98 \pm 1.21
wpbc	69.96 \pm 10.43	75.57 \pm 9.78	9.44 \pm 0.95	3.68 \pm 1.99
zoo	94.78 \pm 5.09	94.98 \pm 5.31	7.00 \pm 0.00	1.39 \pm 0.49

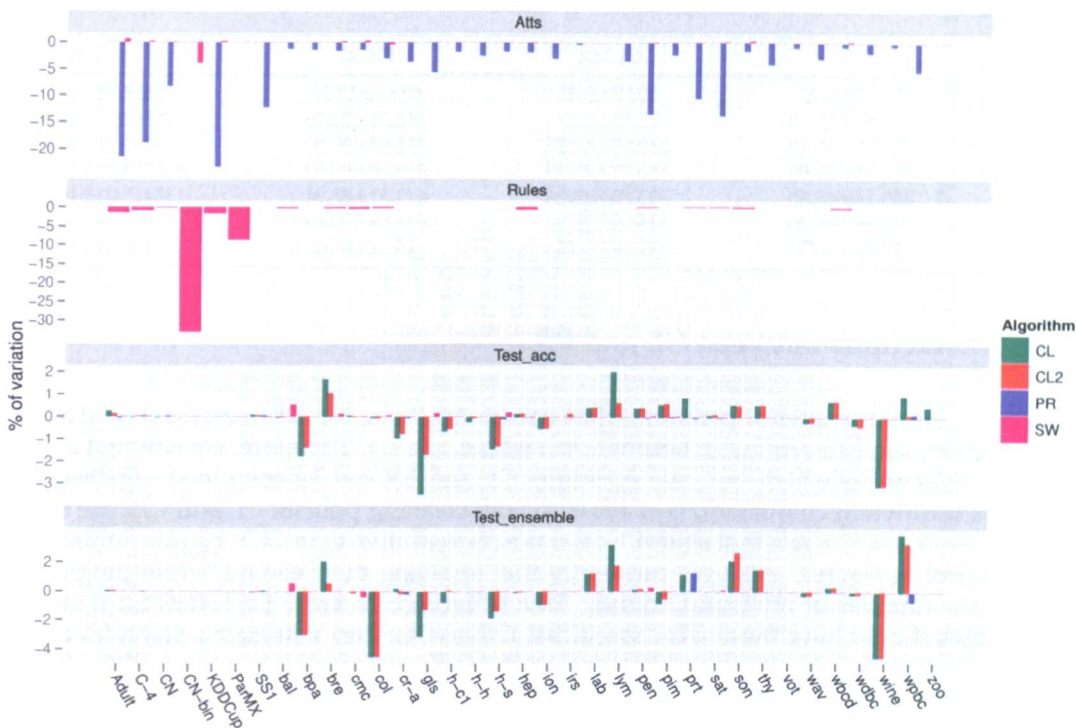


Figure 9.5: Results of the application of the different post-processing operators independently. CL - rule cleaning policy 1, CL2 - rule cleaning policy 2, PR - rule pruning and SW - rule swapping

of the rules in the *CN-bin* problem, and we can observe that it has a bigger impact on the large problems, which had (in general) larger initial rule sets. It is worth noticing that in the *CN-bin* problem, which is the problem where SW erases most rules, the number of attributes also decreases. This indicates that in this particular problem the SW operator managed to keep the most general rules, erasing specific ones that, after changing the order, were not necessary any more. Moreover, considering that *CN-bin* and *ParMX* are binary problems it is expected that the SW operator works better on these domains, because finding similarities between rules in these cases is easier. Finally, the CL operator does not have an effect on the number of attributes or the number of rules by itself, and it concentrates the largest accuracy drops, but as it will be shown later in this chapter, it can boost the performance of the PR operator. We can also observe differences between the two cleaning policies studied. CL2 is much more conservative than CL, and therefore the variations in the accuracy are smaller. In most cases the variations in accuracy seem to depend on the problem rather than on the employed operator.

Table 9.2 shows the total execution times of the CL2, PR and SW operators applied over large problems. The execution times for small problems are less than 1 second for any for the operators and they are not reported. Here we can see that for the three operators the execution time increases proportionally to the size of the training sets and the analysed rule set. Moreover, it is noticeable that the non-conservative cleaning is slightly more expensive than the conservative one. Perhaps because it allows more changes in the predicates. Considering these large datasets, CL, CL2 and PR scale well, but SW involves considerably larger execution times. This is because this operator is very greedy and tries to swap every rule in the solution, recalculating the accuracy each time. Its execution time can be reduced by applying more selective policies which only try to perform a change if the similarity is higher than a threshold. Another way option would be to parallel the execution of this operator using GPGPUs. However, this would not give good results in terms of speedup as the algorithm is mainly sequential. Nevertheless, a different parallel approach to swap the rules could be investigated.

Table 9.2: Execution time of the application of each one of the different operators independently

Prob	CL	CL2 (s)	PR (s)	SW (s)
Adult	50.87±5.02	49.87±3.85	69.60±10.22	5855.04±874.14
C-4	130.87±17.43	96.49±8.33	192.21±24.76	18763.03±2614.41
KDD	311.73±41.19	213.95±18.25	375.85±59.00	23791.21±5041.45
ParMX	526.83±69.04	405.77±37.05	619.20±82.02	106343.70±13094.78
SS1	453.14±45.55	293.70±23.26	649.51±85.94	133415.03±19160.27
CN	446.56±80.20	314.02±26.01	631.68±70.09	43097.44±5429.48
CN-bin	17.54±0.75	17.44±0.76	20.52±0.82	157.51±76.42

9.5.2 Interactions between CL and PR

Considering the nature of the operators, when combining them, the SW operator should always be applied last, since this operator acts over the whole rule set. Therefore, we will first analyse the different ways in which our rule operators CL and PR can be combined together. The results of the different combinations of PR with both cleaning policies CL and CL2 are shown in Figure 9.6. It is noticeable that since CL2 is more conservative than CL it has less impact over the number of attributes. Also we can notice that applying the cleaning before the pruning increases the number of attributes erased. This is because to erase irrelevant attributes the relevant ones should have the correct specificity. Otherwise, they will match examples that do not belong to the domain of that classifier thus making the pruning process more difficult. Also this figure shows that applying pruning twice with CL/CL2 in between, can erase even more attributes than the previous combinations (up to 30%). This is because the PR mechanism can help the CL operator and vice versa in a cyclic way. However, for some problems this involves more variations in the test accuracy, specially in the *gls* and *wine* problems where the accuracy drops 4% and 3%, respectively. It is also worth noticing that when using the conservative CL2 operator, the usage of a previous pruning (PR-CL2-PR) increases dramatically the number of attributes erased, up to a point that for small problems it erases as many attributes as PR-CL-PR. However, the test accuracy stays less variable in PR-CL2-PR. For this reason we consider that the combination PR-CL2-PR could be more appropriate to post-process rule sets.

9.5.3 Interactions of rule operators with SW

Figure 9.7 shows the combinations of the CL, CL2 and PR operators with the rule-set-wise operator SW and also the combination PR-CL2-PR-SW to analyse the behaviour of the three mechanism together under the configuration that provided the best results in the previous section. In this figure we can observe that the number of deleted rules does not change when SW is combined with other rule set operators. The only interesting change is presented in the *CN-bin* problem, where the combination CL2-SW makes the system erase slightly less rules. Moreover, the changes in accuracy remain the same for the CL-SW and CL2-SW operators, showing that SW has no effect over the accuracy changes. In general the SW operator does not affect the performance of the others. Once again the policy that deletes more attributes and more rules is PR-CL2-PR-SW, which erases up to 27% of the attributes or 30% of the rules, depending on the problem. However, this policy drops the test accuracy for some problems, which does not occur with the PR-SW combination. In this sense, we could say that the PR-SW combination is the safest one to use, while the PR-CL2-PR-SW is the best in terms of the obtained results.

9.5.4 Statistical evaluation of the operators

In this section the differences between the original and post-processed rule sets will be studied from a statistical perspective using the Friedman and Holm statistical tests. The tests have been run separately for the average test accuracy, ensemble accuracy, number of rules and number

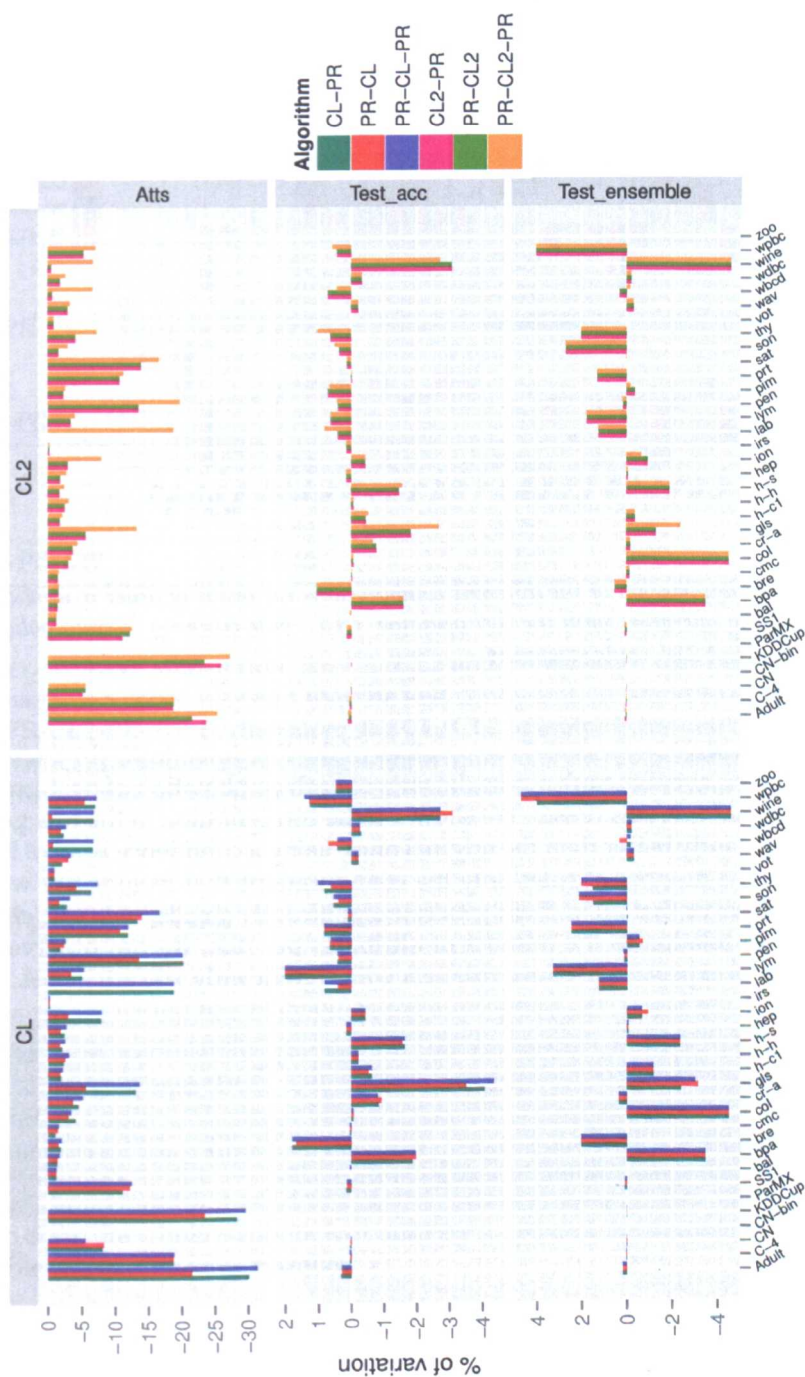


Figure 9.6: Results of the application of the different combinations (in terms of order of application) of post-processing operators CL/CL2 and PR. On the left the combinations with CL (rule cleaning considering the first policy), and on the right the combinations with CL2 (rule cleaning considering the second policy).
PR - rule pruning and SW - rule swapping

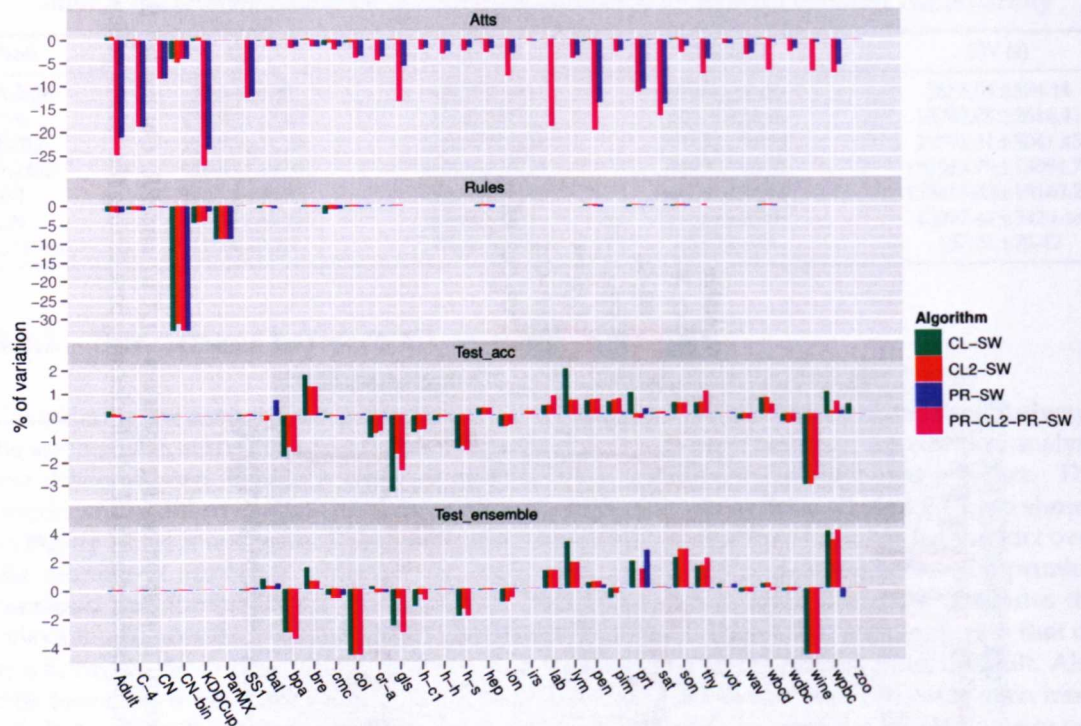


Figure 9.7: Results of the application of the different combinations (in terms of order of application) of post-processing operators with the SW operator. CL2 - rule cleaning (second policy), PR - rule pruning and SW - rule swapping

of attributes in a rule. Table 9.3 reports (a) the p-values of the Friedman tests, (b) the results of the Holm tests and (c) the average rank across datasets of each combination of post-processing operators and the original rule sets (Base). We show * if the configuration in the row generates rule sets that are significantly better than the original ones, using a confidence level of 99%. For the comparison using the number of rules only the configurations that include SW were considered, and we can observe that all of them generate significantly smaller rule sets compared to the original ones. In relation to the number of attributes per rule, we can observe that all settings including PR (as expected) produce rule sets with significantly less attributes.

From the average ranks it is possible to conclude that the best configurations that erase both rules and attributes are PR-CL2-PR-SW and PR-SW. While the first one is capable of erasing more attributes, the second one does not produce major oscillations in the prediction accuracy. Moreover, if the user is not interested in erasing rules but only attributes (and hence avoid the computational cost of SW) the best configurations are PR-CL-PR and PR-CL2-PR.

9.6 Conclusions

In this chapter we have presented three post-processing operators capable of reducing the complexity of rule sets in terms of rule set size and number of attributes expressed per rule. Our results across both small and large problem, show that it is possible to reduce the number of rules and attributes expressed up to 30% in some problems, without significantly degrading the accuracy.

Table 9.3: Rankings of the Friedman statistical tests to compare the combinations of post-processing operators. ★ indicates that the algorithm is significantly better than the initial results according to the Holm test with 99% confidence. For the comparison between the number of rules only the combinations with SW were considered in the statistical test

	Test acc	Test ensem	# Rules		# Atts
P-Values	0.708	0.962	8.9e-09		2.2e-16
Base	7.80	7.07	3.73		10.84
CL	7.73	7.86	–		10.84
CL2	7.64	7.84	–		10.84
PR	7.57	7.21	–		5.53
SW	7.51	6.60	2.59	★	11.30
CL-PR	6.37	7.29	–		3.97
PR-CL	6.67	7.31	–		5.53
PR-CL-PR	5.87	6.79	–		1.51
CL2-PR	6.59	6.79	–		5.81
PR-CL2	6.89	7.16	–		5.71
PR-CL2-PR	6.36	6.91	–		2.29
CL-SW	7.14	6.51	2.07	★	11.23
CL2-SW	7.46	6.83	2.40	★	11.17
PR-SW	6.94	6.29	2.14	★	5.94
PR-CL2-PR-SW	6.46	6.54	2.07	★	2.47

Moreover, we have shown how the CL/CL2 and PR operator can help each other boost their performance in a cyclic way and that the best configuration found to erase both attributes and rules is PR-CL2-PR-SW. Furthermore, the CL/CL2 operator is the one that produces the biggest changes in accuracy, and these changes seem to depend on the problem itself. It is still not clear which characteristic of the problem or the rule sets themselves produces this behaviour.

9.7 Further work

In the future we would like to evaluate the application of CL and PR during the learning process, to see if they could help finding better solutions along the learning process. In the specific case of BioHEL, this means to apply them to each rule learned before inserting them into the iteratively generated rule set. Moreover, we intend to determine why the accuracy drops in some problems when using the CL operator and how to fix this. Also it could be interesting to test other policies for the CL and PR attributes. For example, in the case of PR, we could test a gradual generalisation instead of a drastic one presented in this chapter. This is instead of trying to erase the whole attribute we could try to expand the domain of the attribute as much as possible without making mistakes.

Moreover, other approaches can be studied for the CL and PR operators in which the attributes are not checked iteratively, but the outcome of each possible change is calculated, similar to how it is done in [Bacardit and Krasnogor, 2009b]. After calculating this, it is possible to fix the attributes in an outcome-dependant order, starting from the ones that produce the best outcome (the change makes the rule more general).

For the SW operator it would be interesting to test other metrics or policies to swap rules (different from the similarity). Moreover, a more selective swapping policy could reduce the execution time, by only trying to swap two rules when the similarity is higher than a threshold, avoiding

diminishing returns. This could be done by analysing empirically the relationship between the similarity (or any other metric used to swap rules) and the success in erasing rules after the swap is performed. However, modelling this is not simple as the model would have to take in account the characteristics of all the rules left in the rule set, and not only the two rules the system tries to swap.

Furthermore, it is still necessary to determine systematically how the operators are affected by noise and to compare them to other local search mechanisms such as the ones in present in MPLCS [Bacardit and Krasnogor, 2009b].

Finally, we would like to evaluate these operators on rule sets generated by other systems (beside BioHEL) generating decision lists using GABIL/hyper-rectangle representations to determine (a) if the impact of the post-processing is larger or smaller and (b) determining if there is another combination of operators more suitable to other systems. In addition, we think that these operators can easily be adapted to other representations. In the case of rule cleaning, its application can be translated to other continuous representations by reducing the covered area with the aim of reducing as much as possible the misclassifications. The only domain where this is not possible is the binary domain, since the specificity of the attribute does not vary gradually. The pruning operator can also be translated to other representations by generalising the boundaries (in the case of continuous representations) or setting *don'tcares* (in the case of discrete representations).

In this chapter we have learned that it is possible to fix the possible mistakes made while using the IRL paradigm, if the solution is revisited as a whole. This means we can have a shorter and more general solution if the rules are put in the right order. Moreover, the generality of the individual rules can be increased after the learning process by applying non-expensive operators and without altering the prediction capacity of the system.

Summary, Conclusions and Further Work

This chapter presents a summary of the work conducted in this thesis to improve the efficiency of the BioHEL system when handling large scale domains. It recapitulates the methodologies followed and the conclusions achieved in each chapter. Moreover, this chapter also highlights the contributions made to the area of rule-based EL and the further research directions for BioHEL and the research field in general.

10.1 Summary of the thesis

With the decrease of the storage costs and the developments in technology in the last two decades, humankind is collecting information from all aspects of our society at an increasing speed. However, the capacity of sophisticated methods of performing data mining over large amounts of data at a reasonable speed is still far behind. Therefore, there is a need to push forward the boundaries of the current state-of-the-art mechanisms to make them able to work with larger and more complex domains.

This thesis was focused on improving the efficiency of BioHEL, an EL system which applies the IRL paradigm. In particular we focus on improving the system to handle larger domains in terms of the number of instances available for training and number of attributes or features presented in the problem.

Towards this aim, an exhaustive preliminary experimentation and analysis was carried out in Chapters 4, 5 and 6.

Chapter 4 **Pittsburgh vs. IRL: Identifying future challenges** was particularly focused on determining strengths, weaknesses and domains of competence of both BioHEL and GAssist using a wide variety of real-world problems. Two shared mechanisms between these systems, the default rule and the ILAS windowing scheme, were analysed tangentially to the most influential parameters for BioHEL and GAssist. In the case of GAssist the number of GA iterations and in the case of BioHEL the coverage breakpoint parameter. Also a standard parameter configuration for both algorithms was identified.

Conclusions. This chapter shows that BioHEL is more suitable than GAssist to handle large scale datasets. On the other hand, it also shows that BioHEL is more dependent on the choice of parameters used (specially the coverage breakpoint) to obtain good results. It is shown that this parameter can be very difficult to set up for an end-user and its right setting is determinant for the success of the algorithm. Moreover, the execution time of BioHEL, even though it is lower than that of the other learning approaches which produced similar results, is still high

and dependent on the size of the problem and the number of rules in the final solution. Finally, the number of rules this system generates is higher than GAssist, and in general there is no guarantee that the solutions generated are minimal.

Chapter 5 Parameter impact in BioHEL's fitness function carries out a more in-depth and systematic analysis of the impact of using different coverage breakpoint settings. For this purpose k -DNF boolean functions were used since, as synthetic problems, it is possible to control and adjust their difficulty in regular incremental steps. The two dimensions of difficulty analysed in this chapter are: a) the number of relevant attributes in the terms of the problem (k) and b) the number of terms in the problem (r). These two dimensions generate a third dimension of difficulty which is the overlapping between rules. This chapter analysed the limitations of the BioHEL system in terms of these two dimensions and tries to determine empirically how the coverage breakpoint parameter should be set.

Conclusions. The results of this analysis showed that if the relevant number of attributes of the problem (k) is known it is possible to set the coverage breakpoint accordingly. Moreover, a model of the probability of negative examples in the problem is presented, which can be used to estimate the difficulty of the problem according to the class imbalance.

Chapter 6 Modelling the initialisation stage of BioHEL performed a theoretical analysis of the learning capacity of BioHEL's knowledge representation. This chapter focused on deriving the probability of having a good initial population. This is having individuals that contain information about the solution of the problem and do not make mistakes, and having a population general enough to cover every possible training example. This formulation aimed to: a) understand how the initialisation stage works, b) how the initialisation of the classifiers can be affected by the problem difficulty, and c) understand how some parameters should be set within the system to ensure a good initial population in BioHEL. The models were derived in this chapter for the binary domain and extended afterwards for the χ -ary domain.

Conclusions. The models generated in this chapter show that as k increases and r decreases it becomes more difficult to generate a good initial population. Moreover, the value p (the initialisation probability of GABIL values) should be set according to the k of the problem to increase the chances of generating good individuals. Finally, since these models depend on k and r it might be possible to measure the empirical number of good individuals and obtain estimates of the characteristics of the problem.

Based on the previous conclusions three key enhancements were proposed and developed within the BioHEL system:

- An automated procedure to determine the problem's structure (based on the theoretical models derived) and set the coverage breakpoint parameter accordingly, which is introduced in Chapter 7.
- A fast GPGPU-based parallel evaluation process, introduced in Chapter 8.
- A set of post-processing operators to refine the generality of the decision lists created using IRL, which is introduced in Chapter 9.

Chapter 7 Automatic theory-based adjustment of the coverage breakpoint presents a heuristic approach to set the coverage breakpoint parameter automatically. Since the problem of determining the adequate coverage breakpoint can be translated to determining the characteristics of the problem (k and r), this approach combines the models presented in Chapters 5 and 6 with this purpose. These models, which correspond to the probability of negative examples and the probability of finding good individuals in a population, are used in a reversed way. This means that we can quantify these two values empirically within the problem and determine feasible values for the number of attributes (k) and number of terms (r). Afterwards, using this information and following the conclusions of Chapter 5, it is possible to set the coverage breakpoint parameter accordingly.

Conclusions. This approach is able to find the adequate coverage breakpoint value for a broad set of problems with and without noise. Also the additional effort of the search mechanisms is discussed and the experiments showed that for some problems it can be reduced by more than half by relaxing some constraints in the heuristic. When testing our approach over a large scale bioinformatic problem this mechanism also managed to adapt the coverage breakpoint parameter to the best value found using exhaustive experimentation, reducing the total experimental time by 71%.

Chapter 8 **Fast evaluation process using GPGPUs** explores parallelisation techniques based on GPGPUs to reduce the execution time of BioHEL. Two different strategies with varying degrees of parallelism are proposed and evaluated. We evaluate the parallel fitness computation mechanism first on their own and then integrated into BioHEL's learning process. The interaction between this approach and other efficiency enhancement techniques also existing in BioHEL such as the ILAS windowing scheme are also studied. Last but not least, we presented models of the execution time of our approach based on the number of instances in the training set and number of attributes in the problem.

Conclusions. From the two approaches presented, this chapter shows that the approach with the lower degree of parallelisation produces the best results. This is because this approach balances better the load of each thread and the degree of parallelism. In particular, this approach produced speedups over the CPU-based evaluation of up to 85X using a Tesla C2070 and 52X speedup using a Tesla C1060. Moreover, it was shown that the ILAS windowing can be successfully combined with our approach and the speedups are cumulative. However, the speedups depend on the characteristics of the problem as it was shown by the models. The integration of the GPGPU-based evaluation with BioHEL's learning process obtained important results on the biggest bioinformatic problem where the execution time was reduced from two weeks to eight hours.

In Chapter 9 **Post-processing the final rule sets** we tackle the problem of refining the final solutions of the BioHEL system, which might not be as general as possible due to the usage of the IRL paradigm. In this chapter we introduce three post-processing operators that apply local search to refine the rules: rule cleaning, rule pruning and rule swapping. The two first operators are rule-wise operators which try to reduce the number of attributes in the rules, thus making them more general. On the other hand, the last operator is a rule-set-wise operator that tries to reduce the number of rules in the final rule sets by swapping their order based on similarities. This operator considers that using IRL the rules might be learned in the wrong order which produces larger rule sets.

Conclusions. The rule pruning and the rule cleaning operators combined together managed to reduce the number of attributes in the rules by up to 30% in some problems without reducing significantly the accuracy. Moreover, the rule swapping operator managed to reduce the number of rules up to 30% in real binary problems. This shows that it is possible to improve the generality of the decision lists learned using IRL by revisiting the solution as a whole after the learning process.

By the end of this thesis we have improved the BioHEL system to make it more efficient and usable. In particular the system is now able to automatically adapt the coverage breakpoint parameter for binary problems, which is a very problem dependent parameter, difficult to adjust by hand. Also the system is more efficient in terms of execution time. Both the preliminary experimentation time and the learning time were reduced, by means of adapting parameters automatically and speeding up the evaluation process, respectively. Finally, the system provides solutions that are more compact, general and readable for an end-user. Even though there is still the need to extend the parameter setting heuristic to work with the whole range of domains BioHEL can accept, these improvements have pushed forward some boundaries of the system making it more suitable to solve larger domains.

10.2 Contributions to the area of Evolutionary Learning

This thesis presented the following contributions to the area of rule-based EL:

- An exhaustive analysis over the BioHEL system, on which its domains of competence were determined and guidelines on how to set up the coverage breakpoint where established according to the characteristics of the problem.
- Covering and Schema bounds were derived for a different representation apart from the widely spread ternary representation. This representation corresponds to the GABIL representation within the ALKR context.
- A heuristic to determine the structure of a problem and adapt the coverage breakpoint parameter automatically was proposed.
- A method to speed up the evaluation process using GPGPUs was proposed which can be combined with the ILAS windowing scheme, producing speedups that are cumulative.
- Three operators to post-process the individual rules and rule sets were proposed to generalise the final decision lists obtained.

Moreover, the methodologies presented in this thesis, although focused on the BioHEL system, can be easily extended to other systems. For example, other systems that used the GABIL or the ALKR representation (e.g GAssist, GABIL, HIDER) can use directly the theoretical models presented in Chapter 6, since so far only models have been derived for the ternary representation $\{1, 0, \#\}$. Moreover, the heuristic approach presented in Chapter 7, although its aim is to set up the coverage breakpoint parameter, what it does is to determine the characteristics of the problem. From this perspective this methodology can be beneficial to set up any evolutionary system in advance with a prior knowledge of the problem to solve. Moreover, the two variants of the GPGPU-based fitness function presented in Chapter 8 can be easily applicable to other systems that use a supervised learning approach. Last but not least, the post-processing operators presented in Chapter 9 can be either applicable to other systems that use decision list, or can be applicable during the learning process as extra local search operators to boost the efficiency of the GA.

10.3 Further work

Different future research steps have been underlined in this thesis. We first summarise the future research directions separated per contributions. Afterwards, we discuss the general future directions for BioHEL and for the field of EL in the context of large scale data mining.

10.3.1 Further research directions on BioHEL's experimentation

Still there is the need to analyse in-depth the impact of the ILAS windowing scheme on BioHEL and determine, based on the characteristics of the problem, which are the limits of the usage of this mechanism without impairing the prediction capacity of the system.

Regarding the general performance of BioHEL observed during our preliminary analysis, it is necessary to improve the system to handle data that has a high overlapping, making the system able to distinguish the structure of the problem rather than generalising it. In this sense there is a lot of work from Orriols [Orriols-Puig, 2008] that could be revised and possibly adapted to BioHEL. Even though this work is based on Michigan LCS, the philosophy of using the imbalance ratio to guarantee the reproductive opportunities of the individuals that belong to

the less represented niche could be applied to BioHEL. This is actually easier in supervised learning approaches as the calculation of the imbalance ratio is straight forward.

In general other techniques such as granular clustering [Pedrycz and Bargiela, 2002, 2012], min-max clustering [Bargiela and Pedrycz, 2013; Simpson, 1993], and inclusion/exclusion clustering [Bargiela et al., 2004] seem to hold the key to handle imbalanced data. In these techniques the data is iteratively clustered together in *hyperboxes* (which are very similar to the hyperplane representation of BioHEL). These techniques allow to generate a representation that simply adapts itself to the underlined model of the data (with hyperboxes that can vary their size in different parts of the search space). Considering the similarity of the hyperbox representation and the hyperplane representation it would be very interesting to analyse these techniques in greater depth and try to apply these concepts to BioHEL to improve its performance on imbalanced data.

Moreover, it would be interesting to compare other ML systems against BioHEL using the k-DNF family of problems to see if the behaviours observed in BioHEL also occur in other systems, and which are the limitations of other systems in terms of overlapping and class imbalance.

10.3.2 Further research directions on modelling the behaviour of BioHEL

Regarding the schema and covering bound models developed it is still necessary to adapt these models to continuous inputs. Recent work [Stalph et al., 2012] presents covering bound models for the continuous domain for the XCSF system that can be adapted to BioHEL.

Moreover, it would be ideal to simplify the current models, so they do not depend on characteristics of the problem that are not easy to quantify, and in this sense, improve the applicability of these formulas to determine parameters in the system. Also, in order to use these models to directly set up parameters in the system it is necessary to develop the reproductive opportunity, sustenance and learning time boundaries. In this way there would be a complete theory that fully explains why the BioHEL system works and how it should be set up correctly. Moreover, it would be interesting to determine differences between IRL and Michigan LCS, and model further limitations this paradigm might have beyond the ones already established for traditional LCSs.

10.3.3 Further work on setting the coverage breakpoint automatically

Regarding the coverage breakpoint heuristic it is still necessary to extend this methodology to χ -ary and continuous domains. For this, not only the corresponding models for the representation should be applied, but also the representative processing mechanisms should be revisited since these representations are more complex. In this case both the cleaning and pruning operators explained in Chapter 9 would be necessary to both align the relevant attributes and delete the irrelevant ones. Also in the χ -ary domain a mechanism to discover e (the number of bits activated in a GABIL attribute) would be needed.

Furthermore, it would be interesting to test other selection policies for k in the heuristic, specially if there is not a single cell that has the highest score. Probably in this case it would be better to average the k of the cells that shared the maximum score. Moreover, recycling the representatives observed during the parameter tuning stage would help the populations converge faster. This could lead to the usage of smaller number of iterations in the GA run, which means a smaller execution time.

10.3.4 Further work on speeding up the evaluation process

For the GPGPU-based evaluation process it would be interesting to extend this methodology to work with more than one device at the time. This would increase the scalability of the algorithm when handling larger datasets. Moreover, it would be interesting to apply this methodology to other systems similar to BioHEL and compare the speedups obtained.

Also a more in-depth analysis of what is the right balance between the workload of the kernels and the level of parallelism should be performed to improve the current methodology. Furthermore, to combine the ILAS windowing scheme and the GPGPU evaluation process in the best way, we need to perform both an analysis of the accuracy loss while using ILAS and an analysis of the impact of the ILAS in the workload, which makes the evaluation process run faster.

10.3.5 Further work on refining the final solutions

Regarding the post-processing operators it would be interesting to test variations of the PR operator. For instance, making the attribute more general instead of removing the whole attribute. Also, it would be advised to test other swap policies different than the similarity presented in this thesis. Moreover, since the swapping mechanism is computationally expensive, it would be ideal to test more conservative approaches which only swap the rules if the measure of similarity is above a certain threshold. This threshold could be determined empirically observing which swaps were efficient in terms of erasing rules.

Moreover, it would be necessary to test these operators under controlled noise conditions to determine how they are affected by this factor. Finally, it would be ideal to compare these operators with the memetic operators presented in [Bacardit and Krasnogor, 2009b], which have a very similar nature.

10.3.6 Where to go from here?

Regarding BioHEL there are still a lot of open research directions to follow. First of all, alternatives to the usage of a coverage breakpoint can be explored. Moreover, alternatives to the fitness function in general can be analysed, such as multi objective approaches.

It is also important to continue trying to characterise real problems, as it is the most practical way in which the BioHEL system can be tuned to serve a particular purpose. Using standard configurations the system might not achieve the best possible result but also extensive preliminary experimentation is not a feasible option when handling large domains.

Other methodologies to parallelise BioHEL such as data-intensive computing should be considered. However, the type of code that runs on these architectures is really simple [Owen et al., 2011; Verma et al., 2009], so the current methodologies used would have to be revisited a possibly simplified. In this sense it would be interesting to perform an analysis of speed against learning quality, comparing simple approaches that can run really fast using data-intensive computing and sophisticated approaches that demand larger computational times. In this sense it would be very important to determine which is the middle point between the two that performs better while solving real domains.

In general, future research in the EL field should be focused in trying to tackle the complexities found in real domains (that are not encountered in usual benchmarks or synthetic problems), such as noise, irregular inner structure of the problem, overlapping problem structure, missing data, imbalanced data, among others. Again, in order to do this the characterisation of the problems and their complexity should be performed to select the best algorithm for the task and tailor its configuration accordingly.

There is definitely a long way to go until EL algorithms manage to cope with the volumes of data that industry, medicine, and other areas of our society handle nowadays. However, these algorithms possess two advantages over other sophisticated ML methods that are very important and need to be exploited: the capacity of output logical rules that can be understood and applied by experts and the intrinsic parallelism within the usage of a GA. Moreover, real-world problems should be solved from a practical perspective. This means that when solving this kind of problems the best method is not always the one that provides the most accurate solutions, but the one that produces reliable, general and understandable results in reasonable time. From this perspective algorithms that apply rule-based EL are good candidates to focus the research on to finally bridge the gap between state-of-the-art ML and real-world complex domains.

Results of the parameter sensitivity analysis of BioHEL and GAssist

A.1 Complete results of the parameter sensitivity analysis for BioHEL

Table A.1 shows the results of the test accuracy for small problems using different default class policies and coverage breakpoint settings. Moreover, Table A.2 shows the corresponding train accuracy for the cases presented in the previous tables. Finally, Table A.3 shows the number of rules generated for each scenario.

A.2 Complete results of the parameter sensitivity analysis for GAssist

Table A.4 shows the results of the test accuracy for small problems using different default class policies and coverage breakpoint settings. Moreover, Table A.5 shows the corresponding train accuracy for the cases presented in the previous tables. Finally, Table A.6 shows the number of rules generated for each scenario.

Major			Minor			Dis			Default class policy			Major			Minor			Dis		
			bal												bpa					
0.5	70.25±5.69	-	66.93±6.13	-	66.77±6.06	57.99±1.17	60.33±3.29	60.33±3.29												
0.25	77.60±3.36		80.35±5.48		80.39±4.48	59.69±1.71	66.36±7.20	68.68±5.96												
0.1	82.54±3.01		87.66±3.35	*	87.53±3.77	66.14±7.35	71.03±6.47	69.32±6.53												
0.05	83.34±3.82	*	86.87±3.44		86.90±3.44	64.97±9.81	69.04±5.60	69.62±6.85												
0.01	75.52±5.58		87.35±3.40		85.91±2.77	66.18±7.56	68.43±6.33	69.04±6.10												
			bre												cmc					
0.5	70.30±1.23		72.07±4.30		68.29±5.98	42.70±0.22	42.70±0.22	42.70±0.22												
0.25	70.30±1.23		74.12±7.88		65.34±8.83	43.52±1.79	42.70±0.22	42.70±0.22												
0.1	66.14±12.87		71.37±9.10		69.57±8.31	53.77±3.66	52.42±3.63	53.50±4.21												
0.05	58.04±8.45		68.90±8.50		70.28±9.59	52.62±3.01	54.14±4.24	55.14±3.28												
0.01	55.59±8.36		68.55±10.58		65.35±7.87	48.08±3.76	51.49±3.66	51.07±2.85												
			col												cr-a					
0.5	96.97±2.77		96.48±4.04		96.20±3.66	85.33±4.25	85.48±4.28	85.19±4.10												
0.25	98.08±1.86		97.02±1.54		97.57±1.97	84.18±3.39	83.89±4.48	85.77±4.29												
0.1	98.07±2.63		97.58±2.31		97.57±1.51	85.49±4.79	86.06±3.95	87.07±5.33												
0.05	98.36±1.42		96.46±2.28		97.29±1.83	85.18±4.95	84.04±3.59	84.75±4.90												
0.01	93.73±4.30		95.38±1.82		94.57±3.09	83.02±5.73	82.15±5.54	85.92±3.77												
			gls												h-cl					
0.5	76.00±7.64		69.38±9.42		70.82±8.33	71.17±10.76	71.68±5.33	69.04±5.03												
0.25	72.84±10.52		72.26±9.21		69.07±8.59	78.17±5.46	80.47±4.63	79.24±7.28												
0.1	77.35±12.50		76.84±10.44		76.45±10.07	80.59±6.02	80.54±4.95	79.83±5.27												
0.05	78.59±11.91		75.12±10.99		77.10±8.85	82.23±6.26	80.19±4.14	79.20±4.63												
0.01	76.92±8.86		73.20±8.75		74.71±10.18	79.55±5.29	77.50±4.05	80.54±3.49												
			hep												h-h					
0.5	86.46±10.30		87.08±6.01		91.04±6.81	97.26±3.59	92.56±4.17	97.26±3.59												
0.25	85.83±10.00		88.46±8.96		89.75±6.74	96.58±3.67	94.60±4.27	96.60±3.93												
0.1	88.58±10.09		85.88±8.86		91.75±8.40	96.60±3.93	94.61±4.53	95.26±3.67												
0.05	86.58±9.01		89.75±7.44		89.17±10.25	96.60±3.93	95.27±3.62	97.27±2.69												
0.01	86.50±10.98		89.71±6.04		85.92±10.14	96.25±4.08	95.55±2.89	96.29±5.50												
			h-s												ion					
0.5	78.52±9.21		74.07±10.76		72.96±12.47	89.69±4.50	93.44±3.37	92.30±4.32												
0.25	77.41±7.29		81.48±9.24		78.89±7.21	89.44±4.91	91.99±3.33	92.57±3.19												
0.1	78.15±8.45		80.37±9.41		79.26±7.03	93.35±4.15	92.87±2.43	92.60±3.35												
0.05	78.89±8.20		78.52±8.34		77.41±8.81	93.16±4.15	89.70±3.73	90.86±3.34												
0.01	78.15±9.15		75.93±7.25		80.37±9.41	78.20±7.72	85.13±4.58	91.14±2.59												
			irs												lab					
0.5	94.00±5.84		94.00±5.84		94.67±6.13	98.57±4.52	94.90±8.31	98.57±4.52												
0.25	94.67±5.26		94.67±5.26		96.00±5.62	97.14±6.02	94.90±8.31	94.90±8.31												
0.1	93.33±4.44		92.67±4.92		94.00±4.92	98.57±4.52	94.90±8.31	98.57±4.52												
0.05	92.67±5.84		93.33±5.44		93.33±7.03	95.71±6.90	96.33±7.77	97.14±6.02												
0.01	92.00±7.57		91.33±8.92		91.33±10.45	97.14±6.02	94.90±8.31	97.14±6.02												
			lym												pen					
0.5	85.33±7.95		75.79±13.20		75.79±13.20	95.41±0.69	93.60±1.54	95.36±0.86												
0.25	82.90±12.44		83.53±11.45		84.27±10.65	95.48±0.73	94.00±1.56	95.17±0.93												
0.1	82.06±11.52		84.28±11.15		80.76±12.52	95.18±0.68	93.40±1.52	94.94±0.58												
0.05	81.57±14.19		83.57±11.71		82.10±11.90	98.44±0.58	98.46±0.50	98.51±0.38												
0.01	77.95±15.38		84.06±10.96		77.41±13.51	98.93±0.42	98.97±0.26	98.96±0.29												
			pim												prt					
0.5	65.11±0.78		74.25±4.66		73.99±4.55	42.39±6.91	48.64±6.01	48.72±7.14												
0.25	68.21±3.78		75.16±5.26		75.27±5.14	42.42±6.39	45.95±8.10	48.53±7.40												
0.1	75.52±3.36		74.49±4.52		74.74±7.77	42.35±6.02	48.89±8.00	49.00±8.03												
0.05	71.89±3.55		72.53±5.92		73.45±3.43	45.64±6.54	48.91±6.34	49.83±7.80												
0.01	72.79±5.39		70.60±5.59		74.87±4.58	45.79±7.77	48.64±8.32	48.10±8.22												
			sat												son					
0.5	86.67±1.63	-	83.98±1.32	-	85.56±0.79	56.65±3.87	58.65±5.84	57.59±5.89												
0.25	86.48±1.52		84.26±1.43		85.76±1.30	83.13±10.79	80.72±10.03	79.76±7.93												
0.1	87.69±1.52		88.86±0.96		88.14±1.50	80.93±13.86	81.74±9.85	84.56±8.85												
0.05	90.54±0.98		91.00±0.71		90.99±1.09	80.36±12.66	79.88±8.62	84.65±7.69												
0.01	91.24±0.45	*	91.41±0.69	*	91.70±0.94	74.02±8.12	76.50±8.04	81.77±10.16												
			thy												tot					
0.5	93.05±4.94		94.37±5.75		95.32±4.89	95.85±3.07	96.81±2.42	96.32±2.50												
0.25	92.12±5.71		94.85±5.59		94.87±5.12	96.08±3.11	96.35±2.43	96.78±3.13												
0.1	91.65±5.62		94.42±3.67		94.42±4.31	94.94±3.60	97.25±3.41	96.08±4.11												
0.05	93.48±5.36		93.44±5.55		95.41±6.43	95.17±3.54	97.93±2.55	95.40±3.77												
0.01	93.98±3.02		92.53±7.14		93.05±4.98	95.63±2.79	95.42±3.41	96.09±3.29												
			wav												whrd					
0.5	62.80±1.88		63.58±1.95		63.80±2.78	95.12±2.66	95.14±2.62	95.55±2.48												
0.25	75.54±1.33	-	74.72±1.29	-	78.74±1.20	96.28±2.25	94.72±1.23	96.13±2.35												
0.1	83.24±1.93		83.74±1.72		84.98±1.59	95.56±2.66	95.44±2.29	95.43±1.75												
0.05	84.68±1.95	*	84.78±1.99	*	85.80±1.67	95.71±2.86	94.57±1.76	94.71±2.14												
0.01	84.54±1.65		85.02±1.95	*	85.44±1.68	96.71±2.34	94.57±2.61	96.57±2.63												
			wdbc												wme					
0.5	94.18±3.65		95.97±1.99		93.15±3.02	94.83±5.77	94.90±3.18	94.83±5.77												
0.25	95.96±1.87		96.35±3.74		96.16±3.61	95.42±5.98	95.98±3.82	96.57±1.95												
0.1	96.14±2.45		96.33±2.98		96.51±3.43	94.83±5.77	94.84±4.23	94.80±5.77												
0.05	95.78±2.21		95.46±3.26		96.14±2.41	93.75±6.31	93.72±5.10	94.84±5.77												
0.01	96.66±1.76		93.85±4.08		96.13±2.18	89.30±5.78	92.55±5.58	91.00±4.80												
			wpbc												zwo					
0.5	75.55±6.22		76.00±6.16		76.96±4.88	92.25±7.34	93.98±5.20	93.98±5.20												
0.25	76.46±3.71		79.42±8.55		75.52±9.58	92.25±7.34	94.98±5.31	93.98±5.20												
0.1	77.36±12.27		82.03±8.82		78.03±6.18	92.25±7.34	94.98±5.31	94.98±5.31												
0.05	76.30±12.97		82.55±8.04		78.94±7.07	92.25±7.34	94.98±5.31	92.25±7.34												
0.01	71.07±9.60		78.56±7.60		75.74±10.43	92.25±6.83	94.98±5.31	91.25±6.83												

Table A.1: Test accuracy in BioHEL using different coverage breakpoints and different default class policies. \star is shown next to the control method when significant differences are found (within the same default class) and $-$ is shown next to the methods that are significantly different (worse) than the control method

Default class policy							
Major		Minor	Dis	Major		Minor	Dis
		bal				bpa	
0.5	67.93±0.63	71.63±1.28	71.77±1.21	57.97±0.13	61.16±0.30	61.16±0.30	
0.25	80.60±2.37	88.60±1.55	88.67±1.92	69.41±7.42	85.64±1.42	84.42±2.53	
0.1	90.10±0.91	97.87±0.46	97.26±0.52	97.39±0.44	99.07±0.51	99.55±0.23	
0.05	91.25±0.43	99.59±0.31	98.76±0.46	99.97±0.10	100.00±0.00	100.00±0.00	
0.01	96.34±1.40	99.72±0.19	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	
		brc				cmc	
0.5	70.28±0.14	79.96±3.84	77.47±3.47	42.70±0.02	42.70±0.02	42.70±0.02	
0.25	70.28±0.14	92.27±0.84	82.44±2.26	44.69±1.16	42.72±0.06	42.72±0.04	
0.1	81.85±5.72	98.06±0.32	97.63±0.47	58.36±1.82	56.26±1.99	59.39±2.36	
0.05	97.47±0.67	98.06±0.32	98.06±0.32	72.49±1.32	74.20±2.59	72.11±1.70	
0.01	97.78±0.49	98.06±0.32	98.06±0.32	94.21±0.37	94.94±0.28	95.34±0.29	
		col				cr-a	
0.5	99.76±0.28	98.34±1.24	97.92±1.45	85.68±0.46	85.65±0.48	85.86±0.42	
0.25	99.94±0.19	99.88±0.16	99.91±0.15	89.48±0.80	93.91±2.13	94.38±0.74	
0.1	100.00±0.00	100.00±0.00	100.00±0.00	99.26±0.27	98.86±0.16	98.65±0.29	
0.05	100.00±0.00	100.00±0.00	100.00±0.00	99.98±0.05	100.00±0.00	100.00±0.00	
0.01	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	
		gls				h-cl	
0.5	92.73±0.47	82.51±2.48	83.44±1.70	84.29±10.15	81.25±6.81	77.15±4.43	
0.25	93.98±0.75	86.50±1.77	86.29±1.86	95.34±1.20	96.88±0.71	96.77±0.62	
0.1	99.07±1.04	99.38±0.22	99.38±0.48	100.00±0.00	100.00±0.00	100.00±0.00	
0.05	99.90±0.22	99.95±0.16	99.95±0.16	100.00±0.00	100.00±0.00	100.00±0.00	
0.01	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	
		hep				h-h	
0.5	99.21±0.53	99.43±0.30	98.78±0.76	97.96±0.37	100.00±0.00	97.92±0.41	
0.25	99.14±0.57	100.00±0.00	99.86±0.30	98.90±0.38	100.00±0.00	98.90±0.57	
0.1	100.00±0.00	100.00±0.00	100.00±0.00	99.96±0.12	100.00±0.00	99.96±0.12	
0.05	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	
0.01	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	
		h-s				ion	
0.5	87.86±3.76	84.98±3.51	80.62±3.21	94.90±3.22	97.21±0.61	96.17±0.79	
0.25	95.84±1.32	97.00±0.80	96.71±0.75	95.76±1.39	99.71±0.10	99.53±0.27	
0.1	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	99.94±0.13	99.81±0.16	
0.05	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	
0.01	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	
		ins				lab	
0.5	98.00±0.50	98.00±0.50	98.00±0.50	100.00±0.00	100.00±0.00	100.00±0.00	
0.25	98.22±0.62	98.22±0.62	98.15±0.63	100.00±0.00	100.00±0.00	100.00±0.00	
0.1	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	
0.05	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	
0.01	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	
		hym				pen	
0.5	98.65±0.48	82.43±1.98	82.13±2.13	96.74±0.44	94.88±1.19	96.53±0.23	
0.25	99.85±0.32	99.78±0.36	99.78±0.36	96.75±0.39	95.23±1.29	96.37±0.63	
0.1	100.00±0.00	100.00±0.00	100.00±0.00	96.47±0.52	94.60±1.33	96.23±0.21	
0.05	100.00±0.00	100.00±0.00	100.00±0.00	99.62±0.04	99.65±0.02	99.65±0.03	
0.01	100.00±0.00	100.00±0.00	100.00±0.00	99.99±0.01	99.99±0.01	99.98±0.01	
		pim				pri	
0.5	65.10±0.09	75.22±0.89	75.36±0.92	58.51±6.79	63.53±5.77	64.30±5.86	
0.25	72.73±1.41	85.56±1.39	83.25±0.83	59.37±6.56	63.01±8.04	64.95±5.40	
0.1	86.46±0.84	98.90±0.30	97.74±0.48	58.65±6.78	67.33±4.32	68.07±2.47	
0.05	99.12±0.33	100.00±0.00	99.96±0.07	77.88±2.72	81.42±1.17	81.87±1.12	
0.01	100.00±0.00	100.00±0.00	100.00±0.00	86.32±2.06	88.07±1.44	89.77±1.43	
		sat				son	
0.5	88.95±0.59	85.85±0.38	88.04±0.74	64.00±5.73	63.10±10.47	62.09±9.73	
0.25	88.99±0.53	85.90±0.66	88.04±0.90	100.00±0.00	100.00±0.00	100.00±0.00	
0.1	91.10±0.54	93.10±0.23	91.54±0.41	100.00±0.00	100.00±0.00	100.00±0.00	
0.05	96.48±1.03	97.47±0.12	96.90±0.12	100.00±0.00	100.00±0.00	100.00±0.00	
0.01	100.00±0.01	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	
		thy				vet	
0.5	99.12±0.60	99.95±0.16	99.69±0.44	96.70±0.28	99.21±0.35	96.32±0.28	
0.25	99.02±0.66	100.00±0.00	99.79±0.27	98.77±0.23	99.23±0.24	98.47±0.27	
0.1	100.00±0.00	100.00±0.00	100.00±0.00	99.49±0.27	99.80±0.11	99.54±0.24	
0.05	100.00±0.00	100.00±0.00	100.00±0.00	99.80±0.11	99.77±0.14	99.80±0.11	
0.01	100.00±0.00	100.00±0.00	100.00±0.00	99.80±0.11	99.80±0.11	99.80±0.11	
		wav				wbcd	
0.5	64.86±1.58	65.01±1.10	65.65±2.37	96.47±0.21	99.40±0.24	97.36±0.31	
0.25	78.33±0.51	76.87±0.45	81.74±0.95	97.68±0.19	99.92±0.11	98.92±0.35	
0.1	92.15±0.44	93.62±0.38	96.04±0.18	99.75±0.36	100.00±0.00	99.30±0.35	
0.05	99.33±0.10	99.55±0.09	99.75±0.07	99.86±0.05	100.00±0.00	99.94±0.11	
0.01	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	
		wdbc				wine	
0.5	99.00±0.37	98.71±0.34	96.52±0.46	99.94±0.20	100.00±0.00	100.00±0.00	
0.25	100.00±0.00	99.57±0.15	99.45±0.20	100.00±0.00	100.00±0.00	100.00±0.00	
0.1	100.00±0.00	99.84±0.08	99.82±0.06	100.00±0.00	100.00±0.00	100.00±0.00	
0.05	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	
0.01	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	
		wpbc				zoo	
0.5	78.81±5.85	81.11±5.68	80.88±5.42	100.00±0.00	99.56±0.56	100.00±0.00	
0.25	77.97±5.75	99.94±0.18	99.89±0.24	100.00±0.00	99.56±0.56	100.00±0.00	
0.1	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	99.56±0.56	100.00±0.00	
0.05	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	99.56±0.56	100.00±0.00	
0.01	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	

Table A.2: Train accuracy in BioHEL using different coverage breakpoints and different default class policies

APPENDIX A: RESULTS OF THE PARAMETER SENSITIVITY ANALYSIS OF BIOHEL AND GASSIST

	Major	Minor	Default class policy		Major	Minor	Dis
		bal				bpa	
0.5	1.19±0.74	3.78±1.19	3.48±1.56		0.00±0.00	1.06±0.24	1.03±0.18
0.25	8.14±2.22	12.72±1.43	11.12±1.70		1.82±2.22	9.24±1.48	8.20±1.48
0.1	18.53±3.04	32.86±2.40	29.18±1.99		14.92±1.51	18.66±1.04	19.48±1.52
0.05	31.04±1.88	49.24±2.89	44.28±3.40		20.05±1.29	23.34±1.34	25.86±1.90
0.01	52.66±10.79	54.28±3.45	64.98±3.49		30.51±1.89	35.59±1.88	48.64±3.32
		bre				cmc	
0.5	0.00±0.00	4.02±1.09	2.27±0.98		0.00±0.00	1.20±0.40	0.18±0.41
0.25	0.00±0.00	10.14±0.73	3.90±0.86		2.19±2.40	1.36±0.48	0.43±0.56
0.1	4.42±4.37	17.26±1.08	21.24±1.39		6.14±5.23	11.84±5.84	12.46±5.92
0.05	16.73±1.39	18.11±1.27	25.90±1.70		34.63±14.56	46.93±13.47	40.98±17.28
0.01	21.01±1.77	29.52±1.67	41.70±2.60		138.92±7.39	174.53±6.29	215.38±5.75
		col				cr-a	
0.5	3.83±0.57	2.85±0.76	3.14±0.37		1.00±0.00	1.03±0.17	1.85±0.36
0.25	3.94±0.55	4.61±0.79	4.49±0.61		2.85±0.90	6.16±3.49	9.67±1.01
0.1	4.30±0.83	4.81±0.62	5.30±0.74		18.11±1.34	16.80±0.98	19.17±1.45
0.05	4.68±0.90	6.68±1.13	7.06±1.45		23.59±1.46	24.83±1.56	28.34±2.32
0.01	10.88±2.39	10.76±3.20	15.43±3.59		31.93±1.84	36.07±1.92	51.05±3.27
		gle				h<l	
0.5	10.64±1.11	7.78±0.93	7.24±0.67		4.24±2.46	4.43±1.15	2.17±0.42
0.25	11.13±1.11	10.03±1.00	8.81±1.13		7.69±1.25	8.77±0.72	8.86±0.76
0.1	14.31±1.17	16.75±1.28	16.27±1.24		12.39±0.89	11.87±1.09	12.27±1.13
0.05	15.71±1.28	20.17±1.75	20.03±1.73		13.26±0.89	12.62±1.03	13.93±1.23
0.01	24.81±2.06	35.27±2.63	34.98±2.74		20.26±1.27	22.61±1.56	30.36±2.54
		hep				h-h	
0.5	3.10±0.56	3.72±0.54	2.46±0.50		1.00±0.00	3.74±0.58	1.88±0.33
0.25	3.10±0.54	4.52±0.60	3.78±0.88		2.40±0.83	3.85±0.60	2.90±0.67
0.1	3.75±0.56	5.38±0.59	4.83±0.79		3.82±0.54	3.84±0.61	4.70±0.81
0.05	3.84±0.62	7.26±0.85	5.98±1.35		4.10±0.45	4.64±0.54	5.65±0.88
0.01	6.41±0.89	11.56±2.05	9.07±2.01		7.71±1.11	5.89±1.10	9.41±1.92
		h-s				ion	
0.5	3.28±2.26	4.42±1.12	2.47±0.52		4.79±2.20	2.52±0.64	1.22±0.45
0.25	8.37±1.32	8.86±0.99	8.57±0.87		6.86±1.32	4.17±0.69	3.14±0.88
0.1	12.33±0.96	12.63±1.24	13.29±1.24		8.98±0.69	5.05±0.63	4.14±0.87
0.05	13.20±1.15	13.47±1.39	14.77±1.51		9.05±0.76	7.87±0.66	10.02±1.37
0.01	18.86±1.33	22.92±1.60	29.69±2.66		12.22±1.05	17.51±2.07	19.78±2.23
		irs				lab	
0.5	2.00±0.06	2.00±0.00	2.02±0.13		1.00±0.00	1.00±0.00	1.00±0.00
0.25	2.58±1.08	2.58±1.04	2.44±0.97		1.00±0.00	1.00±0.00	1.00±0.00
0.1	5.45±0.78	5.44±0.81	5.44±0.84		1.00±0.00	1.39±0.49	1.00±0.00
0.05	5.92±0.88	5.85±0.90	6.06±0.92		1.40±0.49	1.68±0.70	1.38±0.52
0.01	10.02±1.64	9.97±1.70	10.56±1.53		1.88±0.57	1.77±0.74	1.88±0.66
		lym				pen	
0.5	6.02±0.54	3.62±0.69	3.48±0.52		34.50±4.31	31.92±3.76	27.49±2.72
0.25	6.80±0.52	8.65±0.90	8.60±0.93		34.41±4.23	32.21±3.57	27.68±2.62
0.1	7.09±0.99	8.86±1.01	8.68±1.03		33.35±3.94	30.91±3.57	26.76±2.77
0.05	7.40±0.59	9.74±1.19	9.35±1.24		54.97±3.03	54.90±3.07	51.42±2.98
0.01	12.32±1.22	18.17±2.22	17.46±2.21		89.07±3.11	88.08±2.84	91.51±2.77
		pim				pri	
0.5	0.00±0.00	1.03±0.20	1.55±0.64		18.50±6.51	22.13±4.46	21.28±4.46
0.25	1.00±0.06	8.89±4.01	6.86±2.25		18.64±6.38	21.96±4.96	21.46±5.22
0.1	6.74±3.99	26.11±1.01	23.57±1.94		18.48±7.19	27.80±4.44	27.06±4.34
0.05	25.91±1.11	34.35±1.63	34.92±1.97		36.88±6.53	45.89±4.80	46.56±4.87
0.01	43.31±2.30	53.91±2.51	66.53±3.48		73.43±6.19	89.72±4.51	89.18±4.68
		sat				sun	
0.5	21.58±4.65	11.05±5.39	16.60±1.77		1.86±0.49	1.66±0.85	1.38±0.73
0.25	21.94±4.67	10.55±5.09	16.58±1.85		5.24±0.61	5.36±0.62	5.65±0.64
0.1	32.68±3.42	37.16±10.17	30.25±2.33		6.16±0.68	5.94±0.68	6.28±0.68
0.05	60.74±5.18	77.60±2.35	65.78±2.97		6.63±0.69	6.90±0.64	8.02±1.05
0.01	152.24±3.87	170.78±3.94	171.97±3.95		13.83±1.47	14.07±1.54	16.28±2.08
		thy				vot	
0.5	4.73±0.82	3.95±0.22	3.67±0.81		2.41±0.57	5.14±0.99	1.00±0.00
0.25	4.69±0.82	3.92±0.27	3.80±0.89		5.14±0.57	5.34±0.55	6.06±1.45
0.1	5.71±0.51	5.49±0.73	6.26±0.89		7.16±0.85	8.38±0.80	12.66±2.11
0.05	5.93±0.41	7.58±0.95	8.66±1.11		8.84±1.40	8.58±0.86	14.37±2.35
0.01	8.49±0.83	10.69±1.32	11.76±1.83		11.79±1.28	11.52±1.38	20.38±2.90
		wav				wbcd	
0.5	7.22±2.53	6.18±3.04	4.43±1.56		3.06±0.39	4.98±0.74	2.74±0.63
0.25	7.65±3.69	3.44±0.94	8.84±1.03		4.46±0.56	6.19±0.65	4.49±1.19
0.1	34.68±2.50	32.03±5.97	32.44±1.57		8.27±1.05	7.00±0.78	5.89±1.33
0.05	62.94±1.45	62.79±1.40	58.89±1.28		8.81±0.74	8.22±0.81	11.01±1.24
0.01	124.06±2.94	122.67±2.99	122.78±3.00		14.49±1.48	14.24±2.00	19.16±2.31
		wdbc				wine	
0.5	4.70±0.75	3.50±0.55	2.72±0.66		2.04±0.19	2.90±0.31	2.01±0.11
0.25	5.46±0.61	4.93±0.62	4.43±0.92		2.03±0.17	2.92±0.34	2.66±0.50
0.1	6.54±0.68	5.95±0.71	6.64±0.87		2.42±0.62	2.98±0.45	2.69±0.63
0.05	6.96±0.73	8.61±0.84	10.02±1.11		3.88±0.66	4.87±0.71	4.34±1.03
0.01	12.18±1.32	17.05±1.87	21.00±2.70		8.40±1.47	8.10±1.35	9.16±1.78
		wdbc				zoo	
0.5	0.66±1.43	2.45±1.57	1.92±0.97		6.00±0.00	6.00±0.00	6.00±0.00
0.25	0.48±1.10	7.96±0.74	5.96±1.12		6.00±0.00	6.00±0.00	6.00±0.00
0.1	5.98±0.69	8.62±0.74	8.44±0.98		6.00±0.00	6.00±0.00	6.00±0.00
0.05	7.00±0.86	10.38±0.85	9.40±1.12		6.00±0.06	6.44±0.55	6.03±0.18
0.01	10.30±0.86	20.04±1.83	17.76±2.35		7.23±0.77	8.86±1.25	7.12±0.75

Table A.3: Number of rules generated in BioHEL using different coverage breakpoints and different default class policies

		Default class policy							
		Major	Minor	Dis	Auto	Major	Minor	Dis	Auto
		bal				bpa			
100	82.10±5.00	85.01±3.13	85.17±3.34	82.89±3.71	65.29±11.41	67.83±5.68	64.66±8.60	65.21±9.94	
250	82.22±4.33	83.07±3.07	82.42±3.25	81.01±4.55	65.24±11.69	68.15±7.46	66.36±6.70	64.72±10.94	
500	81.74±3.69	83.50±2.21	84.02±1.92	82.75±3.85	65.81±8.18	67.26±6.95	66.68±8.45	64.73±11.53	
1000	81.12±4.52	84.02±3.19	83.73±2.90	82.43±2.98	63.80±7.76	65.82±8.50	68.44±6.90	65.57±10.21	
1500	81.12±4.06	83.06±2.57	84.48±2.56	83.68±2.76	64.34±6.56	66.68±8.10	64.36±7.89	64.40±9.18	
		bre				cmc			
100	72.43±6.60	74.21±9.75	74.90±8.10	73.45±6.65	54.52±2.85	54.66±3.10	55.47±4.23	54.80±3.42	
250	74.19±6.90	73.87±9.37	74.52±8.99	72.09±5.32	54.66±3.01	54.39±2.76	54.73±4.36	55.40±3.72	
500	73.49±6.95	74.52±8.86	72.10±7.31	71.73±5.49	55.06±3.00	54.25±2.87	54.32±2.88	55.06±2.77	
1000	71.03±7.51	73.86±7.90	73.87±9.30	72.78±5.36	56.16±4.63	53.98±2.82	54.25±3.43	56.09±3.87	
1500	72.09±7.90	72.45±11.00	74.14±10.16	71.74±5.41	56.02±4.70	53.64±1.94	54.73±3.42	55.95±3.32	
		col				cr-a			
100	96.21±2.94	96.46±3.21	96.74±2.54	96.19±2.65	85.04±4.11	86.06±4.17	84.61±4.47	85.62±4.63	
250	96.18±3.25	96.73±3.15	97.00±2.06	97.00±3.06	85.63±3.88	85.33±3.72	84.60±4.78	85.62±3.84	
500	96.21±2.94	97.01±2.42	97.28±2.27	96.46±2.28	85.48±4.51	85.91±4.19	85.62±3.96	85.33±3.77	
1000	95.91±3.26	96.46±3.46	96.45±2.60	96.46±3.21	84.90±4.57	85.48±4.34	85.34±4.46	86.50±4.09	
1500	96.46±3.42	96.47±3.20	95.92±2.67	96.18±1.94	85.49±4.58	85.77±4.31	85.62±4.57	85.91±4.23	
		gls				h-cl			
100	67.03±10.23	66.65±10.62	64.76±10.23	64.74±8.82	81.20±5.59	83.16±5.26	83.11±6.12	83.17±5.42	
250	70.41±9.95	64.79±11.58	64.70±9.05	67.14±10.64	81.47±4.10	82.83±4.87	82.13±5.81	82.16±4.50	
500	70.37±11.26	66.57±8.38	66.61±10.37	66.65±9.72	80.21±4.84	82.80±4.69	82.47±5.51	81.52±4.15	
1000	70.36±8.58	69.02±9.87	69.07±12.30	68.46±11.89	81.17±4.95	82.77±5.02	82.82±5.17	82.81±4.97	
1500	72.19±8.51	67.40±11.35	66.05±10.54	70.80±10.70	79.18±4.97	83.12±5.39	80.48±4.74	83.15±4.25	
		hep				h-h			
100	92.29±6.60	90.33±7.55	92.25±6.59	92.29±7.23	96.26±3.75	97.61±2.87	96.93±3.46	97.27±3.59	
250	93.54±7.38	92.25±6.59	92.92±6.39	90.21±10.85	96.26±3.75	97.61±2.87	97.27±3.59	97.27±3.59	
500	91.50±9.94	90.25±6.59	90.96±7.47	90.96±7.47	96.26±3.75	97.25±2.76	97.27±3.59	97.24±3.26	
1000	90.92±7.61	91.00±7.41	93.54±6.10	92.25±6.59	96.60±3.93	96.58±2.34	96.93±3.46	96.59±3.93	
1500	92.25±7.95	90.38±6.80	92.88±5.66	92.29±5.00	96.60±3.93	96.24±2.59	97.27±3.59	97.28±3.14	
		h-s				ion			
100	80.37±8.56	83.33±7.86	82.59±8.20	82.96±8.04	93.16±4.32	93.15±3.91	92.28±4.36	93.44±3.37	
250	79.63±8.60	82.59±7.21	82.96±7.45	81.85±8.81	93.16±3.87	93.13±4.73	92.53±4.79	94.29±3.36	
500	79.26±7.24	81.85±7.29	80.74±7.37	80.74±8.15	93.15±3.67	93.42±4.36	93.69±5.09	93.14±4.97	
1000	80.00±6.34	82.22±6.94	83.33±6.36	81.85±7.08	93.15±4.13	94.00±4.19	92.81±5.87	93.15±3.91	
1500	79.26±6.81	81.85±7.29	83.33±6.36	81.48±7.81	92.87±4.99	93.14±5.32	93.69±5.09	94.00±4.45	
		ira				lab			
100	95.33±5.49	95.33±5.49	95.33±5.49	95.33±5.49	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	
250	94.67±5.26	96.00±4.66	94.67±5.26	95.33±5.49	98.57±4.52	100.00±0.00	100.00±0.00	100.00±0.00	
500	95.33±5.49	96.00±4.66	95.33±5.49	96.00±5.62	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	
1000	95.33±5.49	95.33±5.49	94.67±5.26	96.00±5.62	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	
1500	95.33±5.49	94.67±5.26	96.00±4.66	95.33±5.49	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	
		lym				pen			
100	85.50±10.03	83.44±12.84	85.98±11.36	85.45±9.56	78.98±2.40	78.37±2.76	78.93±2.40	76.46±3.16	
250	85.41±11.26	86.07±11.20	85.52±11.19	84.82±11.51	83.35±2.80	83.44±2.80	83.16±2.10	78.63±3.91	
500	86.79±10.25	86.19±10.26	86.12±10.51	86.79±10.25	86.50±1.58	85.94±2.51	85.26±2.50	82.83±2.77	
1000	84.69±12.41	84.02±14.10	82.68±12.19	86.79±10.25	88.70±0.97	88.44±1.71	88.56±1.39	87.27±1.51	
1500	84.03±10.50	85.31±14.06	84.07±12.76	85.41±11.26	89.16±0.98	89.37±1.53	89.06±0.94	88.86±1.33	
		pim				prt			
100	73.99±5.52	73.86±4.57	73.86±4.50	74.64±5.69	51.77±8.49	47.99±6.78	49.41±6.07	47.74±5.28	
250	73.99±5.32	74.77±5.28	74.78±4.61	75.02±4.57	52.79±7.51	51.04±5.52	51.30±6.14	49.75±6.50	
500	75.94±5.21	75.56±4.71	75.95±4.94	75.81±5.00	52.13±7.18	50.99±7.81	51.33±6.22	51.95±4.30	
1000	75.28±3.93	76.47±4.41	75.17±4.96	75.81±5.00	53.61±8.11	53.23±6.19	51.74±6.58	51.71±6.68	
1500	75.28±4.50	76.33±4.60	74.90±5.01	75.42±5.12	54.38±8.26	53.48±6.91	51.70±6.09	51.74±5.77	
		sat				son			
100	82.27±0.89	82.24±1.36	82.59±1.09	82.35±1.27	80.24±8.19	78.86±6.36	77.31±7.51	80.27±6.29	
250	82.73±1.26	83.01±1.26	83.01±1.18	82.95±1.24	81.70±6.39	80.34±6.00	76.04±9.77	81.31±8.04	
500	83.81±1.69	83.50±1.05	83.34±1.22	83.15±1.27	83.65±7.24	77.55±8.28	74.61±7.85	82.67±6.49	
1000	85.53±1.10	83.64±1.15	83.76±1.11	83.74±0.94	86.53±7.13	77.54±7.54	77.95±9.15	82.68±8.66	
1500	86.18±1.26	84.02±1.22	83.70±1.10	84.37±1.25	84.15±9.37	77.91±5.86	73.70±8.75	81.85±8.62	
		thy				vot			
100	93.05±4.45	93.03±4.53	92.58±4.98	93.48±5.01	96.32±4.13	97.47±2.03	97.24±2.13	97.01±2.70	
250	93.03±4.53	92.55±5.89	93.05±5.46	93.51±5.86	96.33±3.98	97.92±2.30	96.54±3.99	97.01±3.30	
500	93.51±3.88	93.48±5.85	92.12±4.89	93.48±5.01	95.40±4.77	97.24±2.13	96.78±3.33	97.47±2.03	
1000	93.03±4.53	92.08±5.39	93.03±4.53	93.51±5.02	94.94±4.88	97.23±2.38	97.01±2.44	97.46±2.30	
1500	93.51±3.88	93.48±5.01	93.51±4.93	92.10±3.82	95.40±4.77	97.46±2.30	96.78±3.33	97.46±2.30	
		wav				wbed			
100	80.92±1.34	81.18±2.05	81.90±2.02	81.62±1.93	96.28±1.54	96.42±2.16	95.85±1.96	96.85±2.11	
250	80.58±2.51	81.40±2.12	82.54±1.69	82.06±1.38	96.42±1.83	95.85±2.38	96.00±2.00	96.56±2.88	
500	81.89±2.83	82.14±1.24	82.36±1.70	82.82±1.61	95.70±1.67	95.99±2.77	96.00±2.22	95.99±2.22	
1000	83.06±2.47	82.70±1.89	83.08±1.87	83.14±1.49	95.84±1.73	95.85±1.59	96.71±2.14	95.71±2.03	
1500	82.86±1.80	83.04±1.74	83.28±2.09	83.24±1.64	95.41±2.03	96.28±2.16	96.13±2.15	96.28±2.16	
		wdbc				wine			
100	95.78±2.39	95.10±3.02	95.61±2.38	95.26±3.11	98.82±2.48	97.68±3.00	96.57±4.83	98.27±2.79	
250	96.31±2.25	95.44±3.31	95.80±3.31	95.79±2.76	96.63±3.91	95.98±4.72	96.04±4.70	96.63±3.91	
500	96.14±2.13	95.63±3.49	94.73±2.89	96.15±2.27	96.63±3.91	96.57±3.95	96.04±3.79	96.04±4.70	
1000	95.97±1.60	95.45±3.29	95.09±3.07	96.14±2.69	96.10±3.76	97.16±3.98	97.16±3.98	96.10±3.76	
1500	95.60±2.41	95.10±4.01	95.26±2.90	95.08±2.96	96.10±3.76	97.22±3.93	96.04±3.79	95.52±3.53	
		wpc				zoo			
100	80.73±8.92	79.54±6.03	80.23±9.14	79.16±7.84	96.35±4.73	94.33±6.54	93.25±8.79	91.36±9.31	
250	82.70±8.68	78.01±6.29	79.08±6.73	82.22±9.36	93.33±6.33	95.98±5.21	95.26±5.02	93.33±6.33	
500	80.81±10.31	78.03±7.43	80.21±9.02	82.70±8.32	94.24±6.59	94.07±5.13	93.53±8.68	94.15±5.09	
1000	82.60±6.89	79.64±8.09	79.66±8.40	81.72±8.31	96.35±4.73	93.24±4.72	94.24±5.01	95.15±5.16	
1500	82.24±9.08	78.08±7.27	79.54±7.60	82.17±7.88	94.33±6.54	94.35±4.89	94.24±5.01	96.98±4.89	

Table A.4: Test accuracy in GAssist using different number of iterations and different default class policies. * is shown next to the control method when significant differences are found (within the same default class) and – is shown next to the methods that are significantly different (worse) than the control method

APPENDIX A: RESULTS OF THE PARAMETER SENSITIVITY ANALYSIS OF BIOHEL AND GASSIST

Default class policy															
Major		Minor		Dis		Auto		Major		Minor		Dis		Auto	
100	85.24±0.40	88.98±0.63	bal	88.75±0.83	88.48±0.95						bpa	76.20±1.39	79.87±2.40		
250	86.40±1.12	89.05±0.73		88.94±0.60	88.39±0.84							79.36±1.58	82.16±2.36		
500	86.83±0.36	89.42±0.61		89.53±0.70	89.19±0.66							82.22±2.07	86.28±2.06		
1000	86.86±1.01	89.65±0.55		89.56±0.63	89.53±0.39							84.41±1.31	87.99±1.61		
1500	87.27±0.72	89.51±0.57		90.03±0.60	89.56±0.55							83.70±1.59	87.99±1.52		
100	82.67±1.15	81.97±1.40	bre	81.27±1.39	82.52±0.95						cmc	57.03±0.59	57.21±0.58		
250	85.00±1.15	85.74±1.23		83.95±1.02	84.81±1.57							57.43±0.59	57.75±0.56		
500	86.67±1.48	88.62±1.10		85.89±1.84	87.30±1.60							58.23±0.45	59.27±0.70		
1000	88.15±1.29	88.93±1.24		87.64±1.79	88.81±1.07							59.55±0.64	61.53±0.93		
1500	89.12±0.92	89.63±1.03		88.42±1.09	89.55±0.96							59.86±0.65	62.04±0.75		
100	99.31±0.85	98.79±1.16	col	98.52±1.32	98.94±0.88						cr-a	88.60±0.88	89.18±0.66		
250	99.85±0.33	99.52±0.79		99.25±0.79	99.64±0.57							89.60±0.81	90.21±0.82		
500	99.91±0.20	99.55±0.26		99.37±0.75	99.85±0.26							90.68±0.50	91.55±0.82		
1000	99.94±0.13	99.64±0.19		99.64±0.58	99.91±0.15							91.43±0.72	92.46±0.66		
1500	99.91±0.20	99.73±0.27		99.61±0.64	99.94±0.13							91.88±0.66	92.77±0.62		
100	77.26±1.69	73.78±2.11	gis	72.54±1.22	72.75±2.01						h-cl	90.72±0.84	90.80±1.20		
250	81.15±1.01	76.65±2.42		76.95±2.20	75.87±2.70							91.97±0.55	92.85±1.04		
500	82.87±1.05	78.87±2.46		78.25±1.93	79.18±1.60							92.63±0.68	94.21±1.29		
1000	83.96±0.72	80.12±1.54		80.22±1.44	80.79±1.13							93.62±0.91	95.34±0.81		
1500	83.39±1.14	81.16±1.39		80.42±0.88	82.71±1.22							93.95±1.10	95.78±0.70		
100	98.78±0.48	98.28±0.37	hep	98.28±0.50	98.92±0.61						h-h	98.64±0.41	98.98±0.51		
250	99.07±0.68	98.21±0.38		98.42±0.57	99.28±0.48							98.64±0.41	99.17±0.35		
500	99.36±0.41	98.42±0.57		98.64±0.41	99.36±0.41							98.71±0.41	99.32±0.30		
1000	99.36±0.41	98.57±0.48		98.64±0.41	99.57±0.50							98.87±0.40	99.47±0.20		
1500	99.43±0.30	98.49±0.53		98.78±0.35	99.50±0.49							98.90±0.38	99.62±0.18		
100	90.25±1.18	90.16±0.79	h-s	89.96±1.18	90.33±0.59						ion	96.04±0.43	96.77±0.43		
250	91.89±0.85	91.89±1.01		91.28±0.72	91.77±1.08							96.30±0.47	97.72±0.33		
500	92.96±1.17	92.26±0.88		91.69±1.08	92.76±0.70							96.68±0.62	98.16±0.29		
1000	93.09±0.80	92.59±0.73		91.77±0.91	93.09±1.09							96.90±0.56	98.19±0.40		
1500	93.13±0.99	93.33±1.06		92.10±1.13	93.50±0.84							97.06±0.40	98.23±0.23		
100	98.37±0.68	98.07±0.52	irs	97.78±0.35	98.22±0.52						lab	100.00±0.00	100.00±0.00		
250	98.37±0.58	98.30±0.61		98.30±0.61	98.30±0.50							100.00±0.00	100.00±0.00		
500	98.52±0.60	98.37±0.68		98.30±0.61	98.30±0.50							100.00±0.00	100.00±0.00		
1000	98.30±0.50	98.22±0.52		98.44±0.55	98.30±0.50							100.00±0.00	100.00±0.00		
1500	98.22±0.52	98.44±0.42		98.22±0.62	98.30±0.50							100.00±0.00	100.00±0.00		
100	97.60±0.68	95.87±1.34	lym	94.67±1.49	95.12±1.22						pen	79.06±1.60	76.54±2.52		
250	98.50±0.79	97.14±1.41		96.25±1.28	97.67±1.09							83.18±2.02	79.03±3.30		
500	99.40±0.47	97.59±1.22		96.84±1.12	97.90±1.16							85.42±1.66	83.07±2.12		
1000	99.63±0.40	98.12±0.88		97.97±1.32	99.10±0.69							88.72±0.43	87.76±0.60		
1500	99.70±0.39	98.43±0.96		98.05±0.95	99.40±0.48							89.43±0.35	89.21±0.50		
100	79.41±0.78	79.72±0.46	pim	78.76±0.97	79.33±0.64						prt	54.48±0.92	50.35±1.89		
250	81.08±0.81	81.21±0.56		81.11±1.04	81.45±0.63							56.50±2.06	52.80±1.83		
500	83.90±0.65	83.09±0.77		82.42±0.62	83.84±0.53							57.93±3.08	56.11±1.62		
1000	84.77±0.78	83.85±0.83		83.06±0.82	84.75±0.76							60.48±4.28	58.19±3.12		
1500	85.05±0.65	84.33±0.53		83.38±0.75	85.01±0.69							62.05±5.24	58.68±3.80		
100	82.75±0.19	82.73±0.23	sat	82.70±0.26	82.72±0.32						smn	92.84±1.80	97.44±0.89		
250	83.15±0.30	83.53±0.13		83.52±0.22	83.40±0.19							96.80±0.71	98.56±0.79		
500	84.25±0.54	83.97±0.15		84.01±0.19	83.72±0.19							97.54±0.80	99.04±0.49		
1000	86.71±0.47	84.36±0.19		84.35±0.14	84.67±0.83							98.18±0.52	99.41±0.47		
1500	86.94±0.40	84.52±0.13		84.46±0.29	85.02±0.72							98.34±0.99	99.63±0.44		
100	98.71±0.61	98.29±0.65	thy	98.04±0.48	98.24±0.50						vot	98.47±0.27	99.05±0.17		
250	99.17±0.50	98.81±0.35		98.45±0.48	98.86±0.59							98.77±0.27	99.11±0.22		
500	99.48±0.49	99.07±0.41		98.66±0.43	99.54±0.45							98.85±0.35	99.11±0.22		
1000	99.54±0.51	99.38±0.47		98.71±0.61	99.64±0.35							99.05±0.17	99.13±0.22		
1500	99.43±0.45	99.23±0.44		99.12±0.49	99.59±0.41							99.03±0.23	99.13±0.22		
100	82.26±0.87	82.27±0.34	wav	83.09±0.51	83.03±0.70						wbcd	97.95±0.26	98.54±0.17		
250	82.54±0.44	82.96±0.38		83.72±0.58	83.81±0.39							98.35±0.28	98.81±0.25		
500	84.34±0.54	84.48±0.48		84.30±0.51	84.55±0.53							98.46±0.39	98.87±0.26		
1000	85.37±0.36	85.57±0.41		85.40±0.56	85.80±0.37							98.59±0.30	98.97±0.13		
1500	85.50±0.37	85.95±0.55		85.38±0.50	85.80±0.25							98.60±0.29	98.95±0.24		
100	97.83±0.39	97.97±0.33	wdbc	97.21±0.43	97.85±0.38						wine	99.94±0.20	99.94±0.20		
250	98.54±0.39	98.77±0.33		97.93±0.29	98.46±0.56							100.00±0.00	100.00±0.00		
500	98.95±0.36	98.98±0.37		98.32±0.51	99.00±0.28							100.00±0.00	100.00±0.00		
1000	99.10±0.31	99.12±0.21		98.48±0.47	99.24±0.21							100.00±0.00	100.00±0.00		
1500	99.14±0.31	99.08±0.23		98.75±0.50	99.34±0.30							100.00±0.00	100.00±0.00		
100	89.79±0.79	86.32±2.38	wpc	85.69±0.92	89.17±1.16						zco	99.56±1.07	98.79±0.97		
250	92.71±1.37	88.89±1.64		87.55±1.27	92.48±0.92							99.67±0.53	99.56±0.78		
500	94.73±1.12	90.29±1.17		88.78±1.02	94.11±1.11							99.55±1.07	99.67±0.54		
1000	95.23±0.74	90.91±1.12		90.18±0.95	94.84±0.93							99.67±0.75	99.78±0.47		
1500	96.63±0.99	91.70±0.93		90.35±1.52	96.07±0.74							99.67±0.75	100.00±0.00		

	Major	Minor	Dis	Default class policy				Major	Minor	Dis	Auto
				Auto	Major	Minor	Dis				
			bal						bpa		
100	4.80±0.80	5.32±0.83	4.22±0.94	4.86±0.88	3.16±0.37	3.12±0.34	3.08±0.28	3.12±0.34			
250	6.79±1.11	6.55±1.14	5.39±1.13	6.40±1.05	4.11±0.95	3.89±0.86	3.38±0.62	4.08±1.00			
500	9.44±1.77	8.36±1.74	7.46±1.65	8.69±1.73	6.48±1.52	5.90±1.42	4.26±1.17	6.50±1.70			
1000	9.52±1.70	8.74±1.60	7.56±1.57	8.98±1.81	7.14±1.73	6.66±1.47	4.43±1.31	7.11±1.83			
1500	9.88±1.82	8.70±1.45	7.76±1.61	8.93±1.72	7.08±1.75	6.75±1.61	4.52±1.23	6.91±1.68			
			bre						cmc		
100	3.94±0.86	3.57±0.66	3.52±0.62	3.76±0.83	3.00±0.00	3.12±0.33	3.03±0.18	3.02±0.13			
250	6.97±1.08	6.20±1.09	4.95±1.31	6.74±1.20	3.14±0.38	3.44±0.63	3.16±0.42	3.15±0.40			
500	8.14±1.35	7.38±1.20	5.94±1.50	8.23±1.41	6.96±1.90	7.54±1.88	5.26±1.60	6.52±1.85			
1000	8.66±1.14	7.73±1.27	6.52±1.43	8.44±1.33	13.75±3.12	13.18±5.41	7.96±3.39	14.16±4.45			
1500	8.93±1.16	8.00±1.25	6.97±1.63	8.76±1.24	14.92±3.60	16.15±6.07	8.28±4.25	16.38±5.90			
			col						cr-a		
100	4.61±1.07	3.78±0.81	3.80±0.93	4.01±1.01	3.03±0.17	3.06±0.23	3.10±0.32	3.03±0.20			
250	5.30±1.25	4.40±1.25	3.98±0.95	4.84±1.28	4.34±1.07	3.94±0.86	3.52±0.73	3.99±0.90			
500	5.36±1.36	4.46±1.45	4.06±1.01	4.80±1.42	6.82±1.27	6.15±1.32	4.48±1.37	6.53±1.38			
1000	5.12±1.27	4.27±1.29	3.94±1.06	4.81±1.36	7.35±1.13	7.00±1.19	4.89±1.36	7.38±1.25			
1500	5.02±1.21	4.21±1.23	3.96±0.98	4.68±1.42	7.54±1.20	7.19±1.06	4.94±1.26	7.34±1.23			
			glc						h-cl		
100	3.86±0.78	4.80±0.90	3.74±0.88	4.18±0.84	4.73±0.89	3.90±0.86	3.55±0.77	4.05±0.89			
250	4.77±0.90	5.21±1.12	4.26±1.05	4.81±1.09	6.75±1.18	5.56±1.07	4.26±1.17	5.88±1.20			
500	5.15±0.99	5.39±1.12	4.36±1.08	5.20±1.32	7.96±1.25	6.35±1.23	4.80±1.42	6.98±1.52			
1000	5.39±0.98	5.36±1.09	4.42±1.00	5.35±1.30	8.20±1.41	6.54±1.21	4.88±1.17	6.89±1.37			
1500	5.32±1.03	5.41±1.19	4.59±1.16	5.48±1.24	7.96±1.27	6.50±1.22	4.84±1.34	6.95±1.42			
			hep						h-h		
100	3.90±0.94	4.58±0.86	3.54±0.77	4.16±0.88	3.21±0.49	3.51±0.65	3.26±0.59	3.33±0.61			
250	3.75±0.82	4.34±0.73	3.42±0.74	4.01±0.90	3.30±0.54	3.72±0.70	3.24±0.58	3.42±0.62			
500	3.82±0.73	4.20±0.76	3.28±0.58	3.92±0.79	3.30±0.56	3.93±0.80	3.20±0.50	3.55±0.69			
1000	3.84±0.78	4.16±0.60	3.24±0.57	3.96±0.66	3.26±0.50	4.11±0.79	3.18±0.42	3.76±0.75			
1500	3.83±0.71	4.07±0.55	3.20±0.45	3.98±0.75	3.30±0.55	4.26±0.70	3.17±0.45	3.75±0.76			
			h-s						ion		
100	4.32±0.84	3.74±0.68	3.35±0.58	4.00±0.78	4.68±1.00	2.79±0.48	3.02±0.33	3.29±1.05			
250	5.84±1.09	4.90±0.94	3.81±0.91	5.29±1.08	5.90±1.26	3.21±0.46	3.08±0.29	3.84±1.37			
500	6.49±1.20	5.42±1.25	4.13±1.23	5.84±1.34	6.68±1.26	3.23±0.57	3.20±0.61	4.11±1.71			
1000	6.35±1.12	5.27±1.05	3.97±1.12	5.74±1.22	6.76±1.20	3.15±0.43	3.06±0.29	3.94±1.64			
1500	6.34±1.09	5.23±1.11	4.06±1.09	5.60±1.21	6.66±1.12	3.10±0.37	3.10±0.38	3.85±1.54			
			irs						lab		
100	3.29±0.52	3.24±0.53	3.20±0.47	3.19±0.52	3.00±0.00	3.00±0.00	3.00±0.00	3.00±0.00			
250	3.22±0.51	3.21±0.47	3.25±0.52	3.14±0.39	3.00±0.00	3.00±0.00	3.00±0.00	3.00±0.00			
500	3.20±0.49	3.16±0.44	3.16±0.41	3.18±0.41	3.00±0.00	3.00±0.00	3.00±0.00	3.00±0.00			
1000	3.16±0.45	3.16±0.43	3.16±0.43	3.16±0.42	3.00±0.00	3.00±0.00	3.00±0.00	3.00±0.00			
1500	3.18±0.44	3.16±0.42	3.16±0.40	3.16±0.44	3.00±0.00	3.00±0.00	3.00±0.00	3.00±0.00			
			lym						pen		
100	6.32±1.15	6.35±1.42	5.44±1.39	5.78±1.38	8.62±1.56	8.65±1.64	8.28±1.78	7.10±1.41			
250	6.71±1.05	6.23±1.27	5.53±1.34	6.36±1.22	8.88±1.57	8.77±1.60	8.39±1.79	7.44±1.22			
500	6.86±1.03	6.24±1.22	5.52±1.38	6.29±1.08	10.60±2.34	10.28±1.99	10.08±2.31	8.86±1.67			
1000	7.08±0.85	6.26±1.12	5.86±1.20	6.42±1.07	13.88±2.83	13.67±3.20	14.18±3.18	12.15±2.28			
1500	6.99±0.81	6.23±1.04	5.75±1.17	6.56±1.07	14.67±3.31	13.98±3.24	14.68±3.20	13.63±2.55			
			pim						pri		
100	3.04±0.20	3.01±0.16	3.00±0.06	3.05±0.23	7.98±1.59	7.69±1.95	6.39±1.76	5.68±1.49			
250	3.73±0.83	3.34±0.63	3.06±0.28	3.63±0.81	10.30±2.33	8.91±2.63	8.05±2.49	6.73±1.76			
500	7.12±1.78	5.80±1.52	3.86±1.13	6.74±1.74	11.96±3.51	11.16±4.34	8.98±3.53	7.54±1.60			
1000	8.27±1.80	6.82±1.71	4.00±1.03	8.05±1.96	14.39±4.54	13.77±5.16	11.12±4.93	8.17±1.73			
1500	8.30±1.74	7.10±1.77	3.94±1.04	7.82±1.90	16.30±4.60	14.58±5.24	11.76±5.09	8.78±1.68			
			sat						son		
100	5.31±1.01	5.74±0.94	4.69±0.93	4.80±0.87	4.38±1.11	4.06±0.98	3.32±0.70	4.06±1.04			
250	5.74±1.11	5.98±1.15	4.66±0.95	4.82±0.86	5.36±1.30	5.20±1.40	3.88±1.06	5.28±1.38			
500	8.78±2.13	7.45±2.51	6.00±1.99	6.33±1.70	5.59±1.36	5.22±1.24	3.91±0.97	5.49±1.42			
1000	13.46±2.83	11.22±3.86	9.68±3.46	10.13±2.31	5.16±1.22	4.76±1.12	3.77±1.00	5.09±1.17			
1500	14.36±3.00	11.64±3.67	9.65±3.23	10.50±2.50	4.80±1.01	4.71±1.21	3.56±0.80	4.85±1.02			
			thy						vot		
100	4.56±0.93	3.58±0.74	3.48±0.71	4.14±0.95	4.10±0.81	4.75±0.68	4.07±0.95	4.56±0.71			
250	4.69±0.97	3.87±0.85	3.52±0.71	4.22±0.93	4.72±0.93	4.91±0.64	4.46±1.08	4.80±0.72			
500	4.71±0.90	3.98±0.87	3.53±0.73	4.30±0.99	4.99±0.91	5.01±0.69	4.58±0.92	4.96±0.74			
1000	4.81±0.85	3.95±0.83	3.44±0.65	4.35±0.90	5.28±0.94	5.04±0.71	4.60±0.85	5.06±0.62			
1500	4.93±0.87	4.04±0.84	3.53±0.70	4.50±0.87	5.29±0.84	5.08±0.69	4.71±0.80	5.09±0.67			
			wav						wbcd		
100	3.24±0.44	3.18±0.38	3.08±0.29	3.08±0.27	4.06±0.65	3.04±0.22	3.08±0.30	3.11±0.38			
250	3.52±0.60	3.36±0.54	3.10±0.31	3.21±0.45	5.10±0.85	3.46±0.64	3.28±0.53	3.64±0.82			
500	6.11±1.65	5.35±1.65	3.63±0.82	4.73±1.50	5.38±0.97	3.70±0.75	3.30±0.63	3.90±1.01			
1000	15.18±4.00	11.74±2.95	8.39±3.29	11.89±3.49	5.38±1.00	3.57±0.74	3.28±0.54	3.77±0.86			
1500	16.08±4.10	12.01±2.97	9.36±3.16	12.58±3.57	5.20±0.90	3.62±0.74	3.25±0.58	3.71±0.88			
			wdbc						wine		
100	3.37±0.57	3.23±0.44	3.14±0.41	3.32±0.52	3.50±0.67	3.85±0.77	3.58±0.82	3.77±0.90			
250	4.34±0.95	3.93±0.79	3.34±0.63	4.09±0.89	3.32±0.53	3.37±0.55	3.24±0.48	3.43±0.71			
500	5.01±1.02	4.26±0.82	3.54±0.79	4.73±1.04	3.13±0.35	3.25±0.48	3.12±0.34	3.44±0.74			
1000	4.97±1.05	4.28±0.80	3.45±0.76	4.70±1.03	3.09±0.37	3.15±0.38	3.11±0.35	3.30±0.63			
1500	4.96±0.90	4.19±0.71	3.54±0.86	4.59±1.06	3.05±0.22	3.12±0.33	3.11±0.31	3.27±0.60			
			wdbc						zoo		
100	3.58±0.70	3.45±0.63	3.10±0.37	3.51±0.68	6.32±0.77	6.44±0.85	6.06±0.98	6.04±0.90			
250	4.49±1.01	4.35±1.00	3.35±0.60	4.45±0.98	6.36±0.74	6.48±0.62	5.95±0.84	6.32±0.81			
500	4.56±1.07	4.23±0.99	3.43±0.75	4.66±1.05	6.41±0.60	6.46±0.68	6.15±0.76	6.32±0.72			
1000	4.46±1.06	4.30±0.96	3.38±0.67	4.53±1.05	6.36±0.58	6.51±0.62	6.16±0.67	6.35±0.60			
1500	4.59±1.14	4.21±1.00	3.39±0.72	4.49±1.01	6.42±0.64	6.51±0.59	6.31±0.70	6.35±0.58			

Table A.6: Number of rules generated in GAssist using different number of iterations and different default class policies

References

- Aguilar-Ruiz, J., Riquelme, J., and Toro, M. (2003). Evolutionary learning of hierarchical decision rules. *IEEE Trans. Syst., Man, Cybern., Part B*, 33(2):324–331.
- Aha, D. W., Kibler, D., and Albert, M. K. (1991). Instance based learning algorithms. *Mach. Learn.*, 6:37–66.
- Alayón, S., Estévez, J. I., Sigut, J., Sánchez, J. L., and Toledo, P. (2006). An evolutionary michigan recurrent fuzzy system for nuclei classification in cytological images using nuclear chromatin distribution. *J. of Biomedical Informatics*, 39:573–588.
- Alcalá-Fdez, J., Sánchez, L., García, S., del Jesus, M. J., Ventura, S., Garrell, J. M., Otero, J., Romero, C., Bacardit, J., Rivas, V. M., Fernández, J. C., and Herrera, F. (2009). Keel: a software tool to assess evolutionary algorithms for data mining problems. *Soft Comput.*, 13:307–318.
- Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Mach. Learn.*, 47(2-3):235–256.
- Bacardit, J. (2004). *Pittsburgh Genetics-Based Machine Learning in the Data Mining era: Representations, generalization, and run-time*. PhD thesis, Ramon Llull University, Barcelona, Spain.
- Bacardit, J. (2005). Analysis of the initialization stage of a pittsburgh approach learning classifier system. In Beyer, H.-G. and O'Reilly, U.-M., editors, *GECCO*, pages 1843–1850. ACM.
- Bacardit, J., Bernadó-Mansilla, E., and Butz, M. V. (2007a). Learning classifier systems: Looking back and glimpsing ahead. In Bacardit, J., Bernadó-Mansilla, E., Butz, M. V., Kovacs, T., Llorà, X., and Takadama, K., editors, *IWLCS*, volume 4998 of *Lecture Notes in Computer Science*, pages 1–21. Springer.
- Bacardit, J., Burke, E. K., and Krasnogor, N. (2009a). Improving the scalability of rule-based evolutionary learning. *Memetic Computing*, 1(1):55–67.
- Bacardit, J. and Butz, M. (2007). Data mining in learning classifier systems: Comparing XCS with GAssist. In Kovacs, T., Llorà, X., Takadama, K., Lanzi, P., Stolzmann, W., and Wilson, S., editors, *Learning Classifier Systems*, volume 4399 of *Lecture Notes in Computer Science*, pages 282–290. Springer Berlin / Heidelberg.
- Bacardit, J. and Garrell, J. M. (2003a). Bloat control and generalization pressure using the minimum description length principle for a pittsburgh approach learning classifier system. In *Proceedings of the 6th International Workshop on Learning Classifier Systems*.
- Bacardit, J. and Garrell, J. M. (2003b). Evolving multiple discretizations with adaptive intervals for a pittsburgh Rule-Based learning classifier system. In *Proceedings of the Genetic and Evolutionary Computation Conference - GECCO2003*, pages 1818–1831. LNCS 2724, Springer-Verlag.
- Bacardit, J., Goldberg, D. E., and Butz, M. V. (2007b). Improving the performance of a pittsburgh learning classifier system using a default rule. In *Learning Classifier Systems, Revised Selected Papers of the International Workshop on Learning Classifier Systems 2003-2005*, pages 291–307. Springer-Verlag, LNCS 4399.

- Bacardit, J., Goldberg, D. E., Butz, M. V., Llorá, X., and Garrell, J. M. (2004). Speeding-Up pittsburgh learning classifier systems: Modeling time and accuracy. In *Parallel Problem Solving from Nature - PPSN VIII*, volume 3242 of *Lecture Notes in Computer Science*, chapter 103, pages 1021–1031. Springer Berlin Heidelberg.
- Bacardit, J., Hirst, J. D., Stout, M., Blazewicz, J., and Krasnogor, N. (2006). Coordination number prediction using learning classifier systems: Performance and interpretability. In *In GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 247–254. ACM Press.
- Bacardit, J. and Krasnogor, N. (2006). Smart crossover operator with multiple parents for a pittsburgh learning classifier system. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 1441–1448, Seattle, Washington, USA. ACM.
- Bacardit, J. and Krasnogor, N. (2008a). Empirical evaluation of ensemble techniques for a pittsburgh learning classifier system. In *Learning Classifier Systems*, volume 4998 of *Lecture Notes on Computer Science*, pages 255–268. Springer Berlin Heidelberg.
- Bacardit, J. and Krasnogor, N. (2008b). Fast rule representation for continuous attributes in genetics-based machine learning. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation - GECCO '08*, page 1421, Atlanta, GA, USA.
- Bacardit, J. and Krasnogor, N. (2009a). A mixed discrete-continuous attribute list representation for large scale classification domains. In *GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 1155–1162, New York, NY, USA. ACM Press.
- Bacardit, J. and Krasnogor, N. (2009b). Performance and efficiency of memetic pittsburgh learning classifier systems. *Evolutionary Computation Journal*, 17(3):in press.
- Bacardit, J. and Llorá, X. (2012). Large scale data mining using genetics-based machine learning. Available at <http://www.slideshare.net/jaumebp/large-scale-data-mining-with-geneticsbased-machine-learning>. Genetic and Evolutionary Computation Conference - GECCO 2012- Tutorial.
- Bacardit, J., Stout, M., Hirst, J. D., Valencia, A., Smith, R., and Krasnogor, N. (2009b). Automated alphabet reduction for protein datasets. *BMC Bioinformatics*, 10(1):6.
- Bacardit, J., Widera, P., Márquez-Chamorro, A., Divina, F., Aguilar-Ruiz, J. S., and Krasnogor, N. (2012). Contact map prediction using a large-scale ensemble of rule sets and the fusion of multiple predicted structural features. *Bioinformatics*, 28(19):2441–2448.
- Bäck, T. (1992). Self-adaptation in genetic algorithms. In *Proceedings of the First European Conference on Artificial Life*, pages 263–271. MIT Press.
- Ball, N. M. and Brunner, R. J. (2010). Data mining and machine learning in astronomy. *International Journal of Modern Physics D*, 19(07):1049–1106.
- Bargiela, A. and Pedrycz, W. (2013). Optimised information abstraction in granular min/max clustering. *Emerging Paradigms in Machine Learning*, pages 31–48.
- Bargiela, A., Pedrycz, W., and Tanaka, M. (2004). An inclusion/exclusion fuzzy hyperbox classifier. *Int. J. Knowledge-based and Intelligent Engineering Systems*, 8(2):91–98.
- Bassel, G. W., Glaab, E., Marquez, J., Holdsworth, M. J., and Bacardit, J. (2011). Functional network construction in arabidopsis using rule-based machine learning on large-scale data sets. *The Plant Cell Online*, 23(9):3101–3116.
- Bauer, E. and Kohavi, R. (1999). An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Mach. Learn.*, 36(1-2):105–139.
- Bernadó-Mansilla, E. and Garrell, J. M. (2003). Accuracy-Based learning classifier systems: Models, analysis and applications to classification tasks. *Evol. Comput.*, 11(3):209–238.

- Bernadó-Mansilla, E., Llorà, X., and Garrell, J. M. (2006). XCS and GALE: a comparative study of two learning classifier systems on data mining. In Lanzi, P., Stolzmann, W., and Wilson, S., editors, *Advances in Learning Classifier Systems*, volume 2321 of *Lecture Notes in Computer Science*, chapter 8, pages 115–132. Springer Berlin / Heidelberg.
- Blake, C., Keogh, E., and Merz, C. (1998). UCI repository of machine learning databases. Available online: www.ics.uci.edu/mllearn/MLRepository.html.
- Breiman, L. (1999). Pasting small votes for classification in large databases and on-line. *Mach. Learn.*, 36(1-2):85–103.
- Browne, W. N. and Ioannides, C. (2007). Investigating scaling of an abstracted LCS utilising ternary and s-expression alphabets. In *Proceedings of the 2007 GECCO conference companion on Genetic and evolutionary computation*, pages 2759–2764, London, United Kingdom. ACM Press.
- Bull, L. (2001). Simple markov models of the genetic algorithm in classifier systems: Multi-step tasks. In *IWLCS '00: Revised Papers from the Third International Workshop on Advances in Learning Classifier Systems*, pages 29–36, London, UK. Springer-Verlag.
- Bull, L. (2002). On using constructivism in neural classifier systems. In *Proceedings of the 7th International Conference on Parallel Problem Solving from Nature, PPSN VII*, pages 558–567, London, UK, UK. Springer-Verlag.
- Bull, L. and Hurst, J. (2000). *Self-Adaptive Mutation in ZCS Controllers*, volume 1803 of *Lecture Notes in Computer Science*, chapter chapter 33, pages 342–349. Springer Berlin Heidelberg.
- Bull, L. and O'Hara, T. (2002). Accuracy-based neuro and neuro-fuzzy classifier systems. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '02*, pages 905–911, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.
- Bull, L., Studley, M., Bagnall, A., and Whitley, I. (2007). Learning classifier system ensembles with rule-sharing. *Evolutionary Computation, IEEE Transactions on*, 11(4):496–502.
- Butz, M., Kovacs, T., Lanzi, P., and Wilson, S. (2004). Toward a theory of generalization and learning in XCS. *Evolutionary Computation, IEEE Transactions on*, 8(1):28–46.
- Butz, M. V. (2005). Kernel-based, ellipsoidal conditions in the real-valued XCS classifier system. In *Proc. Genetic Evol. Comput. Conf., GECCO 2005*, pages 1835–1842, New York, NY, USA. ACM.
- Butz, M. V. (2006). *Rule-Based Evolutionary Online Learning Systems: A Principled Approach to LCS Analysis and Design*, volume 109 of *Studies in Fuzziness and Soft Computing*. Springer.
- Butz, M. V. (2007). Combining gradient-based with evolutionary online learning: An introduction to learning classifier systems. In *HIS '07: Proceedings of the 7th International Conference on Hybrid Intelligent Systems*, pages 12–17, Washington, DC, USA. IEEE Computer Society.
- Butz, M. V., Goldberg, D. E., and Lanzi, P. L. (2005a). Gradient descent methods in learning classifier systems: improving XCS performance in multistep problems. *Evolutionary Computation, IEEE Transactions on*, 9(5):452–473.
- Butz, M. V. and Herbort, O. (2008). Context-dependent predictions and cognitive arm control with XCSF. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation, GECCO '08*, pages 1357–1364, New York, NY, USA. ACM Press.
- Butz, M. V., Lanzi, P. L., Llorà, X., and Loiacono, D. (2008a). An analysis of matching in learning classifier systems. In *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 1349–1356, New York, NY, USA. ACM Press.
- Butz, M. V., Lanzi, P. L., and Wilson, S. W. (2006a). Hyper-ellipsoidal conditions in XCS: rotation, linear approximation, and solution structure. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation, GECCO '06*, pages 1457–1464, New York, NY, USA. ACM.

- Butz, M. V., Lanzi, P. L., and Wilson, S. W. (2008b). Function approximation with XCS: hyperellipsoidal conditions, recursive least squares, and compaction. *Evolutionary Computation, IEEE Transactions on*, 12(3):355–376.
- Butz, M. V., Llorà, X., Pelikan, M., and Goldberg, D. E. (2005b). Extracted global structure makes local building block processing effective. In *XCS. GECCO 2005: Genetic and Evolutionary Computation Conference: Volume*, pages 655–662.
- Butz, M. V., Pedersen, G. K., and Stalph, P. O. (2009). Learning sensorimotor control structures with XCSF: redundancy exploitation and dynamic control. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, GECCO '09, pages 1171–1178, New York, NY, USA. ACM.
- Butz, M. V. and Pelikan, M. (2006). Studying XCS/BOA learning in Boolean functions: structure encoding and random Boolean functions. In *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 1449–456, New York, NY, USA. ACM Press.
- Butz, M. V., Pelikan, M., Llorà, X., and Goldberg, D. E. (2006b). Automated global structure extraction for effective local building block processing in XCS. *Evol. Comput.*, 14(3):345–380.
- Butz, M. V., Stalph, P. O., and Lanzi, P. L. (2008c). Self-adaptive mutation in XCSF. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 1365–1372, Atlanta, GA, USA. ACM Press.
- Cantú-Paz, E. (1998). A survey of parallel genetic algorithms. *Calculateurs Paralleles, Reseaux et Systems Repartis*, 10(2):141–171.
- Cantú-Paz, E. (2000). *Efficient and Accurate Parallel Genetic Algorithms*. Kluwer Academic Publishers, Norwell, MA, USA.
- Carvalho, A. G. and Araujo, A. F. (2009). Improving NSGA-II with an adaptive mutation operator. In *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, pages 2697–2700, Montreal, Québec, Canada. ACM.
- Casillas, J., Carse, B., and Bull, L. (2007). Fuzzy-XCS: A michigan genetic fuzzy system. *IEEE T. Fuzzy Systems*, 15(4):536–550.
- Catanzaro, B., Sundaram, N., and Keutzer, K. (2008). Fast support vector machine training and classification on graphics processors. In *Proceedings of the 25th International Conference on Machine Learning (ICML 2008)*, pages 111, 104.
- Chang, E. (2008). Mining large-scale social networks: Challenges & scalable solutions. Available at <http://mmds.stanford.edu/slides2008/chang.pdf>. Workshop on Algorithms for Modern Massive Data Sets (MMDS).
- Chitty, D. M. (2007). A data parallel approach to genetic programming using programmable graphics hardware. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1566–1573, London, England. ACM.
- Clune, J., Goings, S., Punch, B., and Goodman, E. (2005). Investigations in meta-GAs: panaceas or pipe dreams? In *GECCO '05: Proceedings of the 2005 workshops on Genetic and evolutionary computation*, pages 235–241, New York, NY, USA. ACM Press.
- Cohen, J. (1960). A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20(1):46, 37.
- Cohen, W. W. (1995). Fast effective rule induction. In *Twelfth International Conference on Machine Learning*, pages 115–123. Morgan Kaufmann.

- Cooper, G. F. and Herskovits, E. (1992). A bayesian method for the induction of probabilistic networks from data. *Mach. Learn.*, 9(4):309–347.
- Cramer, N. L. (1985). A representation for the adaptive generation of simple sequential programs. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 183–187, Hillsdale, NJ, USA. L. Erlbaum Associates Inc.
- DaCosta, L., Álvaro Fialho, Schoenauer, M., and Sebag, M. (2008). Adaptive operator selection with dynamic multi-armed bandits. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 913–920, Atlanta, GA, USA. ACM Press.
- Dam, H. H., Abbass, H. A., and Lokan, C. (2005a). Be real! XCS with continuous-valued inputs. In *GECCO '05: Proceedings of the 2005 workshops on Genetic and evolutionary computation*, pages 85–87, New York, NY, USA. ACM.
- Dam, H. H., Abbass, H. A., and Lokan, C. (2005b). DXCS: an XCS system for distributed data mining. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, GECCO '05, pages 1883–1890, New York, NY, USA. ACM.
- Darwin, C. (1909). *The Origin of Species*. Number v. 11 in Harvard classics. P.F. Collier & son.
- Davis, L. (1989). Adapting operator probabilities in genetic algorithms. In *Proceedings of the third international conference on Genetic algorithms*, pages 61–69, George Mason University, United States. Morgan Kaufmann Publishers Inc.
- Davis, L. and Mitchell, M. (1991). Handbook of genetic algorithms. *Van Nostrand Reinhold*.
- De Jong, K. (1988). Learning with genetic algorithms: an overview. *Mach. Learn.*, 3(2-3):121–138.
- Dean, J. and Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113.
- Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2000). A fast elitist Multi-Objective genetic algorithm: NSGA-II. *Evolutionary Computation, IEEE Transactions on*, 6:182–197.
- Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.*, 7:1–30.
- Derrac, J., Garcí̃a, S., and Herrera, F. (2010). Stratified prototype selection based on a steady-state memetic algorithm: a study of scalability. *Memetic Computing*, pages 183–199.
- Dixon, P., Corne, D., and Oates, M. (2003). A Ruleset Reduction Algorithm for the XCS Learning Classifier System. In *Learning Classifier Systems*, volume 2661 of *Lecture Notes in Computer Science*, pages 20–29. Springer Berlin / Heidelberg.
- Dorigo, M. and Colombetti, M. (1997). *Robot Shaping: An Experiment in Behavior Engineering*. Intelligent Robotics and Autonomous Agents. MIT Press.
- Drugowitsch, J. (2008). *Design and Analysis of Learning Classifier Systems: A Probabilistic Approach (Studies in Computational Intelligence)*. Springer Publishing Company, Incorporated, 1 edition.
- Dunn, O. J. (1961). Multiple comparisons among means. *Journal of the American Statistical Association*, 56(293):pp. 52–64.
- Edakunni, N., Kovacs, T., Brown, G., and Marshall, J. A. (2009). Modeling UCS as a mixture of experts. In *GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 1187–1194, New York, NY, USA. ACM.
- Ehrenfeucht, A., Haussler, D., Kearns, M. J., and Valiant, L. (1988). A general lower bound on the number of examples needed for learning. In *Proceedings of the first annual workshop on Computational learning theory*, pages 139–154, MIT, Cambridge, Massachusetts, United States. Morgan Kaufmann Publishers Inc.

- Eiben, A. E., Horvath, M., Kowalczyk, W., and Schut, M. C. (2007a). *Reinforcement Learning for Online Control of Evolutionary Algorithms*, volume 4335 of *Lecture Notes in Computer Science*, chapter chapter 10, pages 151–160. Springer Berlin Heidelberg.
- Eiben, A. E., Michalewicz, Z., Schoenauer, M., and Smith, J. E. (2007b). *Parameter Control in Evolutionary Algorithms*, volume 54 of *Studies in Computational Intelligence*, chapter chapter 2, pages 19–46. Springer Berlin Heidelberg.
- Fernández, A., García, S., Luengo, J., Bernadó-Mansilla, E., and Herrera, F. (2010). Genetics-based machine learning for rule induction: State of the art, taxonomy, and comparative study. *Evolutionary Computation, IEEE Transactions on*, 14(6):913–941.
- Fialho, A., Schoenauer, M., and Sebag, M. (2009). Analysis of adaptive operator selection techniques on the royal road and long k-path problems. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 779–786, Montreal, Québec, Canada. ACM.
- Fogel, L. J. (1964). *On the Organization of Intellect*. PhD thesis, University of California, Los Angeles.
- Franco, M. A., Krasnogor, N., and Bacardit, J. (2010a). Speeding up the evaluation of evolutionary learning systems using GPGPUs. In *GECCO '10: Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 1039–1046, New York, NY, USA. ACM.
- Franco, M. A., Krasnogor, N., and Bacardit, J. (2011). Modelling the initialisation stage of the ALKR representation for discrete domains and GABIL encoding. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation, GECCO '11*, pages 1291–1298, New York, NY, USA. ACM Press.
- Franco, M. A., Krasnogor, N., and Bacardit, J. (2012a). Analysing BioHEL using challenging boolean functions. *Evolutionary Intelligence*, 5:87–102. 10.1007/s12065-012-0080-9.
- Franco, M. A., Krasnogor, N., and Bacardit, J. (2012b). Post-processing operators for decision lists. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference, GECCO '12*, pages 847–854, New York, NY, USA. ACM Press.
- Franco, M. A., Krasnogor, N., and Bacardit, J. (2013a). Auto-tuning a rule-based machine learning algorithm via problem structure identification. *Journal of Machine Learning Research*. Submitted for review on January 2013.
- Franco, M. A., Krasnogor, N., and Bacardit, J. (2013b). GAssist vs. BioHEL: critical assessment of two paradigms of genetics-based machine learning. *Soft Computing*, pages 1–29. Available online: <http://dx.doi.org/10.1007/s00500-013-1016-8>.
- Franco, M. A., Martínez, I., and Gorriñ, C. (2010b). Supply chain management sales using XCSR. In Bacardit, J., Browne, W., Drugowitsch, J., Bernadó-Mansilla, E., and Butz, M., editors, *Learning Classifier Systems*, volume 6471 of *Lecture Notes in Computer Science*, pages 145–165. Springer Berlin / Heidelberg.
- Frank, E. and Witten, I. H. (1998). Generating accurate rule sets without global optimization. In *Proceedings of the Fifteenth International Conference on Machine Learning, ICML '98*, pages 144–151, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.
- Freitas, A. A. (2002). *Data Mining and Knowledge Discovery with Evolutionary Algorithms*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.
- Freund, Y. and Schapire, R. E. (1996). Experiments with a new boosting algorithm. In *Thirteenth International Conference on Machine Learning*, pages 148–156, San Francisco. Morgan Kaufmann.
- Friedman, M. (1937). The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32(200):675–701.

- Fu, C. and Davis, L. (2002). A modified classifier system compaction algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '02*, pages 920–925, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.
- Fürnkranz, J. (1999). Separate-and-conquer rule learning. *Artif. Intell. Rev.*, 13:3–54.
- Galizia, A., Viti, F., Clematis, A., and Milanesi, L. (2010). A dynamic parallel approach to recognize tubular breast cancer for tma image building. In *Proceedings of the 2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing, PDP '10*, pages 403–410, Washington, DC, USA. IEEE Computer Society.
- Gao, Y., Wu, L., and Huang, J. (2006). Ensemble learning classifier system and compact ruleset. In *Simulated Evolution and Learning*, volume 4247 of *Lecture Notes in Computer Science*, pages 42–49. Springer Berlin / Heidelberg.
- García, S., Derrac, J., Cano, J., and Herrera, F. (2012). Prototype selection for nearest neighbor classification: Taxonomy and empirical study. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(3):417–435.
- García, S., Fernández, A., Luengo, J., and Herrera, F. (2009). A study of statistical techniques and performance measures for genetics-based machine learning: accuracy and interpretability. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, 13(10):959–977.
- García, S., Fernández, A., Luengo, J., and Herrera, F. (2010). Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power. *Information Sciences*, 180(10):2044 – 2064. Special Issue on Intelligent Distributed Information Systems.
- García, S. and Herrera, F. (2009). An extension on "statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons. *Journal of Machine Learning Research*, 9:2677–2694.
- Giraldez, R., Aguilar-Ruiz, J. S., and Riquelme, J. C. (2005). Knowledge-based fast evaluation for evolutionary learning. *Trans. Sys. Man Cyber Part C*, 35(2):254–261.
- Glaab, E., Bacardit, J., Garibaldi, J. M., and Krasnogor, N. (2012). Using rule-based machine learning for candidate disease gene prioritization and sample classification of cancer gene expression data. *PLoS ONE*, 7(7):e39932.
- Goldberg, D. E. (1989). *Genetic Algorithms for Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.
- Goldberg, D. E. (2002). *The Design of Innovation: Lessons from and for Competent Genetic Algorithms*. Kluwer Academic Publishers, Norwell, MA, USA.
- Grefenstette, J. (1986). Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems Man and Cybernetics*, 16:122–128.
- Gropp, W., Lusk, E., Doss, N., and Skjellum, A. (1996). A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Comput.*, 22(6):789–828.
- Grossman, R. and Gu, Y. (2008). Data mining using high performance data clouds: experimental studies using sector and sphere. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining, KDD '08*, pages 920–927, New York, NY, USA. ACM.
- Guyon, I. and Elisseeff, A. (2003). An introduction to variable and feature selection. *J. Mach. Learn. Res.*, 3:1157–1182.
- Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. (2009). The WEKA data mining software: an update. *SIGKDD Explor. Newsl.*, 11(1):10–18.

- Harding, S. and Banzhaf, W. (2007). Fast genetic programming on GPUs. In *Proceedings of the 10th European conference on Genetic programming, EuroGP'07*, pages 90–101. Springer-Verlag, Berlin, Heidelberg.
- Harik, G. (1999). Linkage learning via probabilistic modeling in the ECGA. IlliGAL report 99010, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign.
- Harris, M. (2007). Optimizing parallel reduction in CUDA. Available at http://developer.download.nvidia.com/compute/cuda/1_1/Website/projects/reduction/doc/reduction.pdf.
- Hernández-Aguirre, A., Buckles, B. P., and Coello, C. A. C. (2001). On learning $kDNF_n^s$ boolean formulas. In *Evolvable Hardware, NASA/DoD Conference on*, volume 0, page 0240, Los Alamitos, CA, USA. IEEE Computer Society.
- Hirschberg, D. S., Pazzani, M. J., and Ali, K. M. (1994). Average case analysis of k-CNF and k-DNF learning algorithms. In *Proceedings of the workshop on Computational learning theory and natural learning systems (vol. 2) : intersections between theory and experiment*, pages 15–28, Cambridge, MA, USA. MIT Press.
- Hochberg, Y. (1988). A sharper bonferroni procedure for multiple tests of significance. *Biometrika*, 75(4):802, 800.
- Hodges, J. and Lehmann, E. (1962). Rank methods for combination of independent experiments in analysis of variance. *Ann. Math. Statist*, 33:482–497.
- Holland, J. (1975). *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. University of Michigan Press.
- Holland, J. H. (1986a). Escaping Brittleness: The possibilities of General-Purpose Learning Algorithms Applied to Parallel Rule-Based Systems. In *Machine learning, an artificial intelligence approach. Volume II*, pages 593–623.
- Holland, J. H. (1986b). A mathematical framework for studying learning in classifier systems. *Phys. D*, 2(1-3):307–317.
- Holland, J. H. and Reitman, J. S. (1978). Cognitive systems based on adaptive algorithms. In Hayes-Roth, D. and Waterman, F., editors, *Pattern-directed Inference Systems*, pages 313–329. Academic Press, New York.
- Holm, S. (1979). A simple sequentially rejective multiple test procedure. *Scand. J. Statist.*, 6(2):65–70. ArticleType: primary_article / Full publication date: 1979 / Copyright ©1979 Board of the Foundation of the Scandinavian Journal of Statistics.
- Holmes, J. H. (1998). Discovering risk of disease with a learning classifier system. In *Proceedings of the 7th International Conference on Genetic Algorithms (ICGA97)*, pages 426–433. Morgan Kaufmann.
- Holmes, J. H., Lanzi, P. L., Stolzmann, W., and Wilson, S. W. (2002). Learning classifier systems: New models, successful applications. *Information Processing Letters*, 82(1):23 – 30. Evolutionary Computation.
- Hoos, H. H. (2012). Programming by optimization. *Commun. ACM*, 55(2):70–80.
- Howard, G. D., Bull, L., and Lanzi, P. L. (2012). A spiking neural learning classifier system. *CoRR*, abs/1201.3249.
- Hurst, J. and Bull, L. (2001a). Self-Adaptation in classifier system controllers. *Artificial Life and Robotics*, 5:109–119.
- Hurst, J. and Bull, L. (2001b). *A Self-Adaptive Classifier System*, volume 1996 of *Lecture Notes in Computer Science*, chapter chapter 6, pages 70–79. Springer Berlin Heidelberg.

- Hurst, J. and Bull, L. (2002). A self-adaptive XCS. In Lanzi, P., Stolzmann, W., and Wilson, S., editors, *Advances in Learning Classifier Systems*, volume 2321 of *Lecture Notes in Computer Science*, pages 333–360. Springer Berlin / Heidelberg. 10.1007/3-540-48104-4_5.
- Hurst, J. and Bull, L. (2006). A neural learning classifier system with self-adaptive constructivism for mobile robot control. *Artificial Life*, 12:353–380.
- Huy, N. Q., Soon, O. Y., Hiot, L. M., and Krasnogor, N. (2009). Adaptive cellular memetic algorithms. *Evolutionary Computation*, 17(2):231–256.
- Ioannides, C., Barrett, G., and Eder, K. (2011). XCS cannot learn all boolean functions. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation, GECCO '11*, pages 1283–1290, New York, NY, USA. ACM Press.
- Ishibuchi, H., Mihara, S., and Nojima, Y. (2012). Parallel distributed hybrid fuzzy gbml models with rule set migration and training data rotation. *Fuzzy Systems, IEEE Transactions on*, PP(99):1.
- Janikow, C. Z. (1993). A knowledge-intensive genetic algorithm for supervised learning. *Mach. Learn.*, 13(2-3):189–228.
- Jin, Y. (2005). A comprehensive survey of fitness approximation in evolutionary computation. *Soft Comput.*, 9(1):3–12.
- John, G. and Langley, P. (1995). Estimating continuous distributions in bayesian classifiers. In *In Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 338–345. Morgan Kaufmann.
- Jong, K. D. and Spears, W. M. (1991). Learning concept classification rules using genetic algorithms. In *Proceedings of the 12th international joint conference on Artificial intelligence - Volume 2*, pages 651–656, Sydney, New South Wales, Australia. Morgan Kaufmann Publishers Inc.
- Kearns, M. J. (1990). *The Computational Complexity of Machine Learning*. MIT Press, Cambridge, Massachusetts.
- Koh, H., Tan, G., et al. (2011). Data mining applications in healthcare. *Journal of Healthcare Information Management*, 19(2):65.
- Kovacs, T. (1997). Xcs classifier system reliably evolves accurate, complete and minimal representations for boolean functions. *COGNITIVE SCIENCE RESEARCH PAPERS-UNIVERSITY OF BIRMINGHAM CSR*.
- Kovacs, T. (2004). *Strength or accuracy: credit assignment in learning classifier systems*. Springer.
- Kovacs, T. (2011). Genetics-based machine learning. In Rozenberg, G., Bäck, T., and Kok, J., editors, *Handbook of Natural Computing: Theory, Experiments, and Applications*. Springer Verlag.
- Koza, J. R. (1992). *Genetic programming: on the programming of computers by means of natural selection*. MIT Press, Cambridge, MA, USA.
- Krasnogor, N. (2004). Self generating metaheuristics in bioinformatics: The proteins structure comparison case. *Genetic Programming and Evolvable Machines*, 5(2):181–201.
- Krasnogor, N. and Gustafson, S. (2004). A study on the use of “self-generation” in memetic algorithms. *Natural Computing*, 3(1):53–76.
- Krasnogor, N. and Smith, J. (2000). A memetic algorithm with self-adaptive local search: TSP as a case study. In Whitley, L. D., Goldberg, D. E., Cantú-Paz, E., Spector, L., Parmee, I. C., and Beyer, H.-G., editors, *GECCO*, pages 987–994. Morgan Kaufmann.
- Krasnogor, N. and Smith, J. (2001). Emergence of profitable search strategies based on a simple inheritance mechanism. In *International Genetic and Evolutionary Computation Conference (GECCO2001)*, pages 432–439, San Francisco, CA. Morgan Kaufmann Publishers.

- Krasnogor, N. and Smith, J. (2005). A tutorial for competent memetic algorithms: model, taxonomy and design issues. *IEEE Transactions on Evolutionary Computation*, 9(5):474–488.
- Kruskal, W. and Wallis, W. (1952). Use of ranks in one-criterion variance analysis. *Journal of the American statistical Association*, 47(260):583–621.
- Lam, C. (2010). *Hadoop in Action*. Manning Pubs Co Series. Manning Publications.
- Langdon, W. B. and Harrison, A. P. (2008). GP on SPMD parallel graphics hardware for mega bioinformatics data mining. *Soft Comput.*, 12(12):1169–1183.
- Langley, P. (1996). *Elements of Machine Learning*. Morgan Kaufmann Series in Machine Learning. Morgan Kaufmann.
- Lanzi, P. L. (2008). Learning classifier systems: then and now. *Evolutionary Intelligence*, 1(1):63–82.
- Lanzi, P. L. and Loiacono, D. (2010). Speeding up matching in learning classifier systems using CUDA. In Bacardit, J., Browne, W., Drugowitsch, J., Bernadó-Mansilla, E., and Butz, M., editors, *Learning Classifier Systems*, volume 6471 of *Lecture Notes in Computer Science*, pages 1–20. Springer Berlin / Heidelberg. 10.1007/978-3-642-17508-4_1.
- Lanzi, P. L., Loiacono, D., Lanzi, P. L., and Loiacono, D. (2007). Classifier systems that compute action mappings. In *In GECCO '07: Proceedings of the 9th annual Conference on Genetic and Evolutionary Computation*, pages 1822 – 1829. ACM Press.
- Lanzi, P. L. and Perrucci, A. (1999a). Extending the representation of classifier conditions part I: from binary to messy coding. In Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, M., and Smith, R. E., editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1, pages 345–352, Orlando, Florida, USA. Morgan Kaufmann.
- Lanzi, P. L. and Perrucci, A. (1999b). Extending the representation of classifier conditions part II: from messy coding to S-Expressions. In Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, M., and Smith, R. E., editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1, pages 345–352, Orlando, Florida, USA. Morgan Kaufmann.
- Lanzi, P. L. and Wilson, S. W. (2006). Using convex hulls to represent classifier conditions. In *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 1481–1488, New York, NY, USA. ACM Press.
- Larragaña, P. and Lozano, J. (2002). *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation*, volume 2 of *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Kluwer Academic Publishers.
- Li, J.-M., Wang, X.-J., He, R.-S., and Chi, Z.-X. (2007). An efficient fine-grained parallel genetic algorithm based on GPU-accelerated. In *Proceedings of the 2007 IFIP International Conference on Network and Parallel Computing Workshops, NPC '07*, pages 855–862, Washington, DC, USA. IEEE Computer Society.
- Lin, M. (2011). Machine learning on big data: Lessons learned from google projects. Guest lecture about machine learning on big data at Harvard on March 29, 2011.
- Liu, B. (2007). *Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data*. Data-Centric Systems and Applications. Springer.
- Llorà, X. (2009). Data-intensive computing for competent genetic algorithms: a pilot study using meandre. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation, GECCO '09*, pages 1387–1394, New York, NY, USA. ACM.
- Llorà, X., Ács, B., Auvil, L. S., Capitanu, B., Welge, M. E., and Goldberg, D. E. (2008). Meandre: Semantic-driven data-intensive flows in the clouds. In *Proceedings of the 2008 Fourth IEEE International Conference on eScience, ESCIENCE '08*, pages 238–245, Washington, DC, USA. IEEE Computer Society.

- Llorà, X. and Garrell, J. M. (2000). Evolving agent aggregates using cellular genetic algorithms. In Whitley, L. D., Goldberg, D. E., Cantú-Paz, E., Spector, L., Parmee, I. C., and Beyer, H.-G., editors, *GECCO*, page 868. Morgan Kaufmann.
- Llorà, X. and Garrell, J. M. (2001). Knowledge-independent data mining with fine-grained parallel evolutionary algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2001)*, pages 461–468. Morgan Kaufmann Publishers.
- Llorà, X., Priya, A., and Bhargava, R. (2009). Observer-invariant histopathology using genetics-based machine learning. *Natural Computing*, 8:101–120. 10.1007/s11047-007-9056-6.
- Llorà, X., Reddy, R., Matesic, B., and Bhargava, R. (2007). Towards better than human capability in diagnosing prostate cancer using infrared spectroscopic imaging. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation, GECCO '07*, pages 2098–2105, New York, NY, USA. ACM Press.
- Llorà, X. and Sastry, K. (2006). Fast rule matching for learning classifier systems via vector instructions. In *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 1513–1520, New York, NY, USA. ACM Press.
- Llorà, X., Sastry, K., and Goldberg, D. E. (2005). The compact classifier system: scalability analysis and first results. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 1, pages 596–603 Vol.1.
- Llorà, X., Sastry, K., and Goldberg, D. E. (2007). Binary rule encoding schemes: A study using the compact classifier system. In Kovacs, T., Llorà, X., Takadama, K., Lanzi, P., Stolzmann, W., and Wilson, S., editors, *Learning Classifier Systems*, volume 4399 of *Lecture Notes in Computer Science*, pages 40–58. Springer Berlin / Heidelberg.
- Llorà, X., Sastry, K., Goldberg, D. E., and delaOssa, L. (2006). The x-ary extended compact classifier system: Linkage learning in pittsburgh LCS. In *Proceedings of the 9th International Workshop on Learning Classifier Systems - IWLCS2006*. (in press), LNAI, Springer-Verlag.
- Llorà, X., Sastry, K., Yu, T., and Goldberg, D. E. (2007). Do not match, inherit: fitness surrogates for genetics-based machine learning techniques. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1798–1805, New York, NY, USA. ACM.
- Llorà, X., Verma, A., Campbell, R., and Goldberg, D. (2010). When huge is routine: Scaling genetic algorithms and estimation of distribution algorithms via data-intensive computing. In Vega, F., Fernández, and Cantú-Paz, E., editors, *Parallel and Distributed Computational Intelligence*, volume 269 of *Studies in Computational Intelligence*, pages 11–41. Springer Berlin Heidelberg.
- Loiacono, D. (2011). Fast prediction computation in learning classifier systems using CUDA. In *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation, GECCO '11*, pages 169–170, New York, NY, USA. ACM.
- Maitre, O., Baumes, L. A., Lachiche, N., Corma, A., and Collet, P. (2009). Coarse grain parallelization of evolutionary algorithms on GPGPU cards with EASEA. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 1403–1410, Montreal, Québec, Canada. ACM.
- Marín-Blázquez, J. G. and Martínez Pérez, G. (2008). Intrusion detection using a linguistic hedged fuzzy-XCS classifier system. *Soft Comput.*, 13(3):273–290.
- McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. In Anderson, J. A. and Rosenfeld, E., editors, *Neurocomputing: foundations of research*, chapter A logical calculus of the ideas immanent in nervous activity, pages 15–27. MIT Press, Cambridge, MA, USA.

- Mellor, D. (2005). A first order logic classifier system. In *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 1819–1826, New York, NY, USA. ACM Press.
- Merz, C. J. (1999). Using correspondence analysis to combine classifiers. *Mach. Learn.*, 36(1-2):33–58.
- Michalski, R., Carbonell, J., and Mitchell, T. (1985). *Machine Learning: An Artificial Intelligence Approach*. Number v. 1 in *Machine Learning: An Artificial Intelligence Approach*. M. Kaufmann.
- Middleton, A. M. (2011). Introduction to HPCC (high-performance computing cluster).
- Mitchell, T. (1997). *Machine Learning*. McGraw-Hill Series in Computer Science. McGraw-Hill Companies, Incorporated.
- Müller, S., Schraudolph, N. N., and Koumoutsakos, P. D. (2002). Step size adaptation in evolution strategies using reinforcement learning. In *Proceedings of the 2002 IEEE Congress on Evolutionary Computation*, pages 151–156. IEEE Press.
- Musilek, P., Li, S., and Wyard-Scott, L. (2005). Enhanced learning classifier system for robot navigation. In *Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on*, pages 3390 – 3395.
- Nemenyi, P. (1963). *Distribution-free multiple comparisons*. PhD thesis, Princeton University, USA.
- NVIDIA (2008). *NVIDIA CUDA Programming Guide 2.0*.
- Orriols-Puig, A. (2008). *New Challenges in Learning Classifier Systems: Mining Rarities and Evolving Fuzzy Models*. PhD thesis, Universitat Ramon Llull, Barcelona, Catalonia, Spain, Barcelona.
- Orriols-Puig, A. and Bernadó-Mansilla, E. (2004). Analysis of reduction for algorithms XCS classifier system. In *in Setè Congrés Català d’Intel·ligència Artificial (CCIA), Frontiers of Artificial Intelligence and Applications*, pages 383–390. IOS Press.
- Orriols-Puig, A. and Bernadó-Mansilla, E. (2008a). Evolutionary rule-based systems for imbalanced data sets. *Soft Comput.*, 13(3):213–225.
- Orriols-Puig, A. and Bernadó-Mansilla, E. (2008b). Revisiting UCS: description, fitness sharing, and comparison with XCS. In *Learning Classifier Systems*, volume 4998 of *Lecture Notes on Computer Science*, pages 96–116. Springer Berlin Heidelberg.
- Orriols-Puig, A., Bernadó-Mansilla, E., Goldberg, D. E., Sastry, K., and Lanzi, P. L. (2009). Facetwise analysis of XCS for problems with class imbalances. *Trans. Evol. Comp*, 13(5):1093–1119.
- Orriols-Puig, A., Casillas, J., and Bernadó-Mansilla, E. (2008a). A comparative study of several genetic-based classifiers in supervised learning. In *Learning Classifier Systems in Data Mining*, volume 125 of *Studies in Computational Intelligence*, chapter 10, pages 205–230. Springer Berlin Heidelberg.
- Orriols-Puig, A., Casillas, J., and Bernadó-Mansilla, E. (2008b). Genetic-based machine learning systems are competitive for pattern recognition. *Evolutionary Intelligence*, 1:209–232.
- Orriols-Puig, A., Casillas, J., and Bernadó-Mansilla, E. (2009). Fuzzy-ucs: a michigan-style learning fuzzy-classifier system for supervised learning. *Trans. Evol. Comp*, 13(2):260–283.
- Orriols-Puig, A., Llorà, X., and Goldberg, D. E. (2010). How XCS deals with rarities in domains with continuous attributes. In *GECCO'10*, pages 1023–1030.

- Orriols-Puig, A., Sastry, K., Goldberg, D., and Bernadó-Mansilla, E. (2008c). Substructural surrogates for learning decomposable classification problems. In Bacardit, J., Bernadó-Mansilla, E., Butz, M., Kovacs, T., Llorà, X., and Takadama, K., editors, *Learning Classifier Systems*, volume 4998 of *Lecture Notes in Computer Science*, pages 235–254. Springer Berlin / Heidelberg. 10.1007/978-3-540-88138-4_14.
- Owen, S., Anil, R., Dunning, T., and Friedman, E. (2011). *Mahout in Action*. In Action Series. Manning Publications.
- Pedrycz, W. and Bargiela, A. (2002). Granular clustering: a granular signature of data. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 32(2):212–224.
- Pedrycz, W. and Bargiela, A. (2012). An optimization of allocation of information granularity in the interpretation of data structures: Toward granular fuzzy clustering. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 42(3):582–590.
- Pelikan, M., Goldberg, D. E., and Cantu-Paz, E. (1999). BOA: The bayesian optimization algorithm. In Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, M., and Smith, R. E., editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1, pages 525–532, Orlando, Florida, USA. Morgan Kaufmann.
- Pettinger, J. E. and Everson, R. M. (2002). Controlling genetic algorithms with reinforcement learning. In *Proceedings of the Genetic and Evolutionary Computation Conference: Morgan Kaufman Publishers Inc.*
- Phua, C., Lee, V. C. S., Smith-Miles, K., and Gayler, R. W. (2010). A comprehensive survey of data mining-based fraud detection research. *CoRR*, abs/1009.6119.
- Platt, J. C. (1999). *Fast training of support vector machines using sequential minimal optimization*, pages 185–208. MIT Press, Cambridge, MA, USA.
- Pospichal, P., Jaros, J., and Schwarz, J. (2010). Parallel genetic algorithm on the CUDA architecture. In *Proceedings of the 2010 international conference on Applications of Evolutionary Computation - Volume Part I, EvoApplications'10*, pages 442–451, Berlin, Heidelberg. Springer-Verlag.
- Prabhu, R. D. (2008). SOMGPU: an unsupervised pattern classifier on graphical processing unit. In *IEEE Congress on Evolutionary Computation*, pages 1011–1018.
- Quade, D. (1979). Using weighted rankings in the analysis of complete blocks with additive block effects. *Journal of the American Statistical Association*, 74(367):pp. 680–683.
- Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
- Rechenberg, I. (1973). *Evolutionsstrategie: optimierung technischer systeme nach prinzipien der biologischen evolution*. Frommann-Holzboog.
- Rissanen, J. (1978). Modeling by shortest data description. *Automatica*, vol. 14:465–471.
- Rivest, R. L. (1987). Learning decision lists. *Machine Learning*, 2(3):229–246.
- Rokach, L. (2010). Ensemble-based classifiers. *Artif. Intell. Rev.*, 33(1-2):1–39.
- Sakurai, Y., Takada, K., Kawabe, T., and Tsuruta, S. (2010). A method to control parameters of evolutionary algorithms by using reinforcement learning. pages 74–79. IEEE.
- Sanders, J. and Kandrot, E. (2010). *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Addison-Wesley.
- Scarpino, M. (2011). *OpenCL in Action: How to Accelerate Graphics and Computations*. Manning Publications.

- Schwefel, H. (1975). *Evolutionsstrategie und numerische Optimierung*. Dr.-Ing. thesis, Technical University of Berlin, Department of Process Engineering.
- Shafi, K. and Abbass, H. A. (2009). An adaptive genetic-based signature learning system for intrusion detection. *Expert Syst. Appl.*, 36(10):12036–12043.
- Sharp, T. (2008). Implementing decision trees and forests on a GPU. In Forsyth, D., Torr, P., and Zisserman, A., editors, *Computer Vision, ECCV 2008*, volume 5305 of *Lecture Notes in Computer Science*, pages 595–608. Springer Berlin Heidelberg.
- Shi, Y., Ye, D., Goder, A., and Narayanan, S. (2011). A large scale machine learning system for recommending heterogeneous content in social networks. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval, SIGIR '11*, pages 1337–1338, New York, NY, USA. ACM.
- Simpson, P. (1993). Fuzzy min-max neural networks– part 2: Clustering. *IEEE Transactions on Fuzzy systems*, 1(1):32.
- Smith, J. and Fogarty, T. (1996a). Self adaptation of mutation rates in a steady state genetic algorithm. In *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on*, pages 318–323.
- Smith, J. and Fogarty, T. C. (1996b). *Adaptively parameterised evolutionary systems: Self adaptive recombination and mutation in a genetic algorithm*, volume 1141 of *Lecture Notes in Computer Science*, chapter chapter 45, pages 441–450. Springer-Verlag.
- Smith, R., Jiang, M., Bacardit, J., Stout, M., Krasnogor, N., and Hirst, J. (2010). A learning classifier system with mutual-information-based fitness. *Evolutionary Intelligence*, 3(1):31–50.
- Smith, S. F. (1980). *A learning system based on genetic adaptive algorithms*. PhD thesis, University of Pittsburgh.
- Smith, S. F. (1983). Flexible learning of problem solving heuristics through adaptive search. In *Proceedings of the Eighth international joint conference on Artificial intelligence - Volume 1*, pages 422–425, Karlsruhe, West Germany. Morgan Kaufmann Publishers Inc.
- Song, D., Heywood, M. I., and Zincir-Heywood, A. N. (2005). Training genetic programming on half a million patterns: an example from anomaly detection. *Trans. Evol. Comp.*, 9(3):225–239.
- Spears, W. M. (1995). Adapting crossover in evolutionary algorithms. *Proceeding of the 4th annual conference on Evolutionary Programming*, pages 367–384.
- Stahl, F., Bramer, M., and Adda, M. (2010). J-pmcri: A methodology for inducing pre-pruned modular classification rules. In *Artificial Intelligence in Theory and Practice III*, volume 331 of *IFIP Advances in Information and Communication Technology*, pages 47–56. Springer Berlin Heidelberg.
- Stalpf, P. O., Butz, M. V., and Pedersen, G. K. M. (2009). Controlling a four degree of freedom arm in 3d using the XCSF learning classifier system. In *Proceedings of the 32nd annual German conference on Advances in artificial intelligence, KI'09*, pages 193–200, Berlin, Heidelberg. Springer-Verlag.
- Stalpf, P. O., Llorà, X., Goldberg, D. E., and Butz, M. V. (2012). Resource management and scalability of the XCSF learning classifier system. *Theoretical Computer Science*, 425:126–141.
- Steel, R. G. D. (1959). A multiple comparison sign test: Treatments versus control. *Journal of the American Statistical Association*, 54(288):pp. 767–775.
- Steinkraus, D., Buck, I., and Simard, P. (2005). Using GPUs for machine learning algorithms. In *Document Analysis and Recognition, 2005. Proceedings. Eighth International Conference on*, pages 1115–1120 Vol. 2.

- Stone, C. and Bull, L. (2003). For real! XCS with continuous-valued inputs. *Evol. Comput.*, 11:299–336.
- Stone, C. and Bull, L. (2008). Foreign exchange trading using a learning classifier system. In Bull, L., Bernadó-Mansilla, E., and Holmes, J. H., editors, *Learning Classifier Systems in Data Mining*, volume 125 of *Studies in Computational Intelligence*, pages 169–189. Springer Berlin / Heidelberg.
- Stout, M., Bacardit, J., Hirst, J. D., and Krasnogor, N. (2008). Prediction of recursive convex hull class assignments for protein residues. *Bioinformatics*, 24(7):916–923.
- Stout, M., Bacardit, J., Hirst, J. D., Smith, R. E., and Krasnogor, N. (2009). Prediction of topological contacts in proteins using learning classifier systems. *Soft Comput.*, 13:245–258.
- Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Mach. Learn.*, 3(1):9–44.
- Tabacman, M., Bacardit, J., Loiseau, I., and Krasnogor, N. (2008). Learning classifier systems in optimisation problems: a case study on fractal travelling salesman problems. In *Proceedings of the International Workshop on Learning Classifier Systems*, volume (to appear) of *Lecture Notes in Computer Science*. Springer.
- TheEconomist (2010a). Data, data everywhere. Available at <http://www.economist.com/node/15579717>. Feb. 25th, 2010.
- TheEconomist (2010b). The data deluge. Available at <http://www.economist.com/node/15557443>. Feb. 25th, 2010.
- Tran, H. T., Sanza, C., Duthen, Y., and Nguyen, T. D. (2007). XCSF with computed continuous action. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation, GECCO '07*, pages 1861–1869, New York, NY, USA. ACM.
- Urbanowicz, R. and Moore, J. (2010). The application of pittsburgh-style learning classifier systems to address genetic heterogeneity and epistasis in association studies. In Schaefer, R., Cotta, C., Kolodziej, J., and Rudolph, G., editors, *Parallel Problem Solving from Nature D PPSN XI*, volume 6238 of *Lecture Notes in Computer Science*, chapter 41, pages 404–413. Springer Berlin Heidelberg.
- Urbanowicz, R. J. and Moore, J. H. (2009). Learning classifier systems: a complete introduction, review, and roadmap. *Journal of Artificial Evolution and Applications*, 2009:1–25.
- Vapnik, V. N. (1995). *The nature of statistical learning theory*. Springer-Verlag New York, Inc., New York, NY, USA.
- Venturini, G. (1993). SIA: a supervised inductive algorithm with genetic search for learning attributes based concepts. In Brazdil, P. B., editor, *Machine Learning: ECML-93 - Proceedings of the European Conference on Machine Learning*, pages 280–296. Springer-Verlag.
- Verma, A., Llorà, X., Goldberg, D. E., and Campbell, R. H. (2009). Scaling genetic algorithms using MapReduce. In *Proceedings of the 2009 Ninth International Conference on Intelligent Systems Design and Applications, ISDA '09*, pages 13–18, Washington, DC, USA. IEEE Computer Society.
- Verma, A., Llorà, X., Venkataraman, S., Goldberg, D. E., and Campbell, R. H. (2010). Scaling eCGA model building via data-intensive computing. In *IEEE Congress on Evolutionary Computation*, pages 1–8. IEEE.
- Wang, Z., Crammer, K., and Vucetic, S. (2012). Breaking the curse of kernelization: Budgeted stochastic gradient descent for large-scale svm training. *Journal of Machine Learning Research*, 13:3103–3131.

- Watkins, C. (1989). *Learning from Delayed Rewards*. PhD thesis, University of Cambridge, England.
- Weiss, S. and Indurkha, N. (1998). *Predictive Data Mining: A Practical Guide*. The Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann Publishers.
- Wilcoxon, F. (1945). Individual comparisons by ranking methods. *Biometrics*, 1(6):pp. 80–83.
- Wilson, S. W. (1994). ZCS: a zeroth level classifier system. *Evol. Comput.*, 2(1):1–18.
- Wilson, S. W. (1995). Classifier fitness based on accuracy. *Evol. Comput.*, 3(2):149–175.
- Wilson, S. W. (2000). Get real! XCS with Continuous-Valued inputs. In *Learning Classifier Systems, From Foundations to Applications, LNAI-1813*, pages 209—219. Springer-Verlag.
- Wilson, S. W. (2001a). Compact rulesets from XCSI. In *Proceedings of the 4th International Workshop on Learning Classifier Systems - IWLCS2001*, pages 197–210. Springer.
- Wilson, S. W. (2001b). Mining oblique data with XCS. In Luca Lanzi, P., Stolzmann, W., and Wilson, S., editors, *Advances in Learning Classifier Systems*, volume 1996 of *Lecture Notes in Computer Science*, pages 283–290. Springer Berlin / Heidelberg.
- Wilson, S. W. (2002). Classifiers that approximate functions. *Natural Comput.*, 1(2-3):211–234.
- Witten, I. H. and Frank, E. (2005). *Data mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann.
- Yoo, A. and Kaplan, I. (2009). Evaluating use of data flow systems for large graph analysis. In *Proceedings of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers, MTAGS '09*, pages 5:1–5:9, New York, NY, USA. ACM.
- Yu, Q., Chen, C., and Pan, Z. (2005). Parallel genetic algorithms on programmable graphics hardware. In *Proceedings of the First international conference on Advances in Natural Computation - Volume Part III, ICNC'05*, pages 1051–1059, Berlin, Heidelberg. Springer-Verlag.
- Zaki, M. and Ho, C. (2000). *Large-Scale Parallel Data Mining*. Lecture Notes in Computer Science. Springer.